Summer 2014

A Model-Based Approach To System-Of-Systems Engineering Via The Systems Modeling Language

Kevin Hughes Bonanne
Purdue University

Graduate School ETD Form 9
(Revised /)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Kevin H. Bonanne

Entitled
A MODEL-BASED APPROACH TO SYSTEM-OF-SYSTEMS ENGINEERING VIA THE SYSTEMS MODELING LANGUAGE

For the degree of Master of Science in Aeronautics and Astronautics

Is approved by the final examining committee:

Daniel DeLaurentis	William Crossley
Saurabh Bagchi	

To the best of my knowledge and as understood by the student in the *C Disclaimer (Graduate School Form)*, this thesis/dissertation Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Daniel DeLaurentis

Approved by: Tom Shih 06/27/2014

Head of the Graduate Program Date

A MODEL-BASED APPROACH TO SYSTEM-OF-SYSTEMS ENGINEERING VIA THE SYSTEMS MODELING LANGUAGE

A Thesis

Submitted to the Faculty

of

Purdue University

by

Kevin H. Bonanne

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

August 2014

Purdue University

West Lafayette, Indiana

To Tom, Carmella, my friends, and my family

ACKNOWLEDGMENTS

The work in this thesis would not be possible without the guidance of Dr. Daniel DeLaurentis, who has taught me and fostered my development over many years; Dr. William Crossley and Dr. Saurabh Bagchi and Bob Kenley, who served on my committee; Dr. Oleg Sindiy, who mentored me through much of my early career; Dr. Thomas McVittie and Dr. Otfrid Liepack, with whom I worked at JPL; Marc Sarrel, who showed me the delicate intricacies of SysML; Kim Simpson, who has acted as project manager for much of my technical work related to this thesis; and finally my colleagues at Purdue University, especially those in Dr. DeLaurentis' research group, who have furthered my understanding of numerous subjects through technical discussions and their own research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SYMBOLS

P_T	average power transferred
n	number of pulses integrated
τ_p	pulse width
G_T	transmitted gain
λ	wavelength
F_n	receiver noise
L_s	system loss
T_o	operating temperature
k	Boltzmann's constant
σ	cross sectional area perpendicular to the sensor
R	sensor's range from the target
σ_θ	angular standard deviation of measurement
σ_R	range standard deviation of measurement
c	speed of light
P_D	probability of detection
P_{FA}	probability of a false alarm

ABBREVIATIONS

bdd	Block Definition Diagram
BMDS	Ballistic Missile Defense System
C2	Command and Control
CAD	Computer-Aided Design
CFD	Computational Fluid Dynamics
CONOPS	Concept of Operations
FEA	Finite Element Analysis
ibd	Internal Block Diagram
MBSE	Model-Based Systems Engineering
RF	Radio Frequency
SoS	System-of-Systems
SoSE	System-of-Systems Engineering
STK	Satellite Tool Kit

ABSTRACT

Bonanne, Kevin H. M.S., Purdue University, August 2014. A Model-Based Approach to System-of-Systems Engineering via the Systems Modeling Language. Major Professor: Daniel DeLaurentis.

In the field of Systems Engineering, a movement is underway to capture the aspects of a system in a centralized model format instead of various documents. This is the basis of Model Based Systems Engineering (MBSE). In order to better formalize this change, the Systems Modeling Language (SysML) was developed to characterize an ontology for MBSE. Despite the growth of both MBSE practices and SysML tools, they have yet to be rigorously analyzed as to their applicability to the field of System-of-Systems (SoS). This thesis applies SysML to a methodology for System-of-Systems Engineering (SoSE) known as the Wave Model, which focuses on an iterative approach to SoS development. Each applicable step in the Wave Model is performed within SysML. Three different SoS types – directed, acknowledged, and collaborative – are studied within the domain of a distrubuted sensor management problem. As each SoS is established, evaluated, and updated, the applicability of SysML to each step is discussed. It is found that SysML is capable of defining, analyzing, and evolving a SoS via the processes described in the Wave Model. SysML excels at strictly defining and organizing the elements and features of a SoS while requiring more development in the analysis portions of the SoSE process.

1. INTRODUCTION

1.1 Motivation

Systems Engineering (SE) has been around for hundreds of years, but truly began evolving as a field in the 1950s [1]. At this time, the growing space, missile, and nuclear warhead races were requiring a higher quality of systems planning, integration, and testing than in the past. Shortly after SE processes reached the commercial industry. Most notably, the Japanese automobile industry quickly adopted SE planning and design processes to see their vehicles lowering in cost and increasing in reliability. This change saw the growth and eventual dominance of Japanese automobile companies worldwide from the 1970s onward [2].

As systems became increasingly complex and more network-centric, the concept of a System-of-Systems (SoS) arose – encompassing that complex group of distributed, independent, and interacting systems [3]. This classification of a system-of-systems required a methodology to design, manage, and analyze a SoS and thus System-of-Systems Engineering (SoSE) was born [4].

The current state of affairs in SE and SoSE involves a document-centric approach to development of systems [5]. In this process, each phase of system development may require a review (e.g., Preliminary Design Review, Test Readiness Review) and a related document or set of documents. This approach provides structure to the systems engineering process; it is not a bad thing. However, a document-centric approach leads to having various sources of information, which may have conflicting information, and can be tedious to manage. Moving away from a document-centric approach and towards a model-centric approach can be highly beneficial [6], as all information is located in a single-source-of-truth, navigable model. The same information is portrayed, but in a more intelligent manner. This shift towards Model-Based

Systems Engineering (MBSE) echoes the changes from pen-and-paper engineering to the computer-aided design that is ubiquitous today in many engineering disciplines.

As there is a difference in how an engineer designs a system and how an engineer designs a SoS, there is also a difference in how one would model a system versus a SoS. However, few groups applying MBSE have examined its applicability to SoSE (see 2 for literature review). This research rigorously examines the current applicability and future challenges of utilizing MBSE techniques, specifically the Systems Modeling Language, for SoSE.

1.2 System-of-Systems

Systems-of-Systems involve a group of systems collaborating towards a common goal. The constituent systems of a SoS may be of varying levels of complexity with varying degrees of autonomy and pursuant of separate individual goals. However, there must remain a semblance of global desire to reach a designated goal.

Mark Maier establishes that two primary traits are necessary for a group of systems to be established as a SoS: operational independence and managerial independence [3]. *Operational independence* establishes that each system is capable of performing a set of functions without any interaction from the other systems; the systems are not so closely integrated that they are operationally co-dependent. *Managerial independence* states that the systems manage themselves to a purpose that is separate from the overarching purpose of the SoS. For example, a Aegis Combat System may be operated by the US Navy for a specific set of functions, including monitoring an airspace for missiles threats. The system has the authority and ability to perform these functions. However, its role in a SoS, such as the Ballistic Missile Defense System (BMDS), expands upon these functions to serve the need of the SoS by taking information from the Aegis system and utilizing it to influence other systems (e.g., sensors, missile batteries). Since the Aegis system and other systems in the BMDS have the capability to operate and manage themselves independently, the BMDS is considered a SoS [27].

Three other properties identified by Maier and often associated with SoS's, though not necessary for the classification, are geographic distribution, evolutionary behavior, and emergent behavior. *Geographical distribution* states that all systems cannot be collocated. The capability for a SoS to change over time – adding, replacing, and altering its systems and network – is established in its *evolutionary behavior.* The *emergent behavior* property describes the fact that by integrating the constituent systems into a SoS, capabilities and behaviors will arise that are not possible with the systems on their own.

Daniel DeLaurentis [7] identified three implications from the properties identified by Maier. He attributes *heterogeneity* of systems, a *trans-domain* nature, and a *network* to connect the systems to the SoS problem. Of all eight traits, the most important as noticed by Maier and DeLaurentis are managerial and operational independence. Though the other traits may exist, the driving factors in defining a SoS are the capabilities of the constituent systems to exist and act on their own (operational independence) while also fulfill new purposes when interacting (managerial independence) [8].

Four different types of SoS have been identified (especially for defense-related content): virtual, collaborative, acknowledged, and directed [9]. In a virtual SoS, the systems lack any sort of central management or unifying purpose, but voluntarily interact to some effect. Virtual SoS's are not controlled by standard engineering means due to their abstract nature and complexity; thus, this type of SoS is not examined in this thesis. A collaborative SoS keeps the voluntary interaction and establishes rules for interaction and inclusion to the SoS. These rules are often controlled by central members of the SoS and are the primary means of controlling and evolving the SoS. For example, the Internet is the most popular example of a collaborative SoS and operates by a set of rules such as the Internet Protocol. With an acknowledged SoS, there is the addition of a designated systems engineering manager, dedicated resources, and a mechanism for shared governance of the SoS while the constituent systems, ultimately maintain their independence. Finally, a directed SoS is specifically managed for a set of objectives and is centrally managed. Often a directed SoS is nearly indistinguishable from a "complex system" due to their shared monolithic nature. SysML applications to directed, acknowledged, and collaborative SoS's are presented in later chapters.

1.3 Model-Based Systems Engineering via the Systems Modeling Language

MBSE involves capturing the various aspects of an integrated system or system-of-systems and incorporating them into a single model. A model will consist of various graphical viewpoints that describe the characteristics of a system. Many of these viewpoints are commonplace within systems engineering – bearing resemblance to flowcharts, state machine diagrams, and structural or functional decomposition diagrams. Furthermore, the model ideally can be connected to various other computer-based engineering tools (e.g., CAD, CFD, FEA, STK) [10] [11], though this research will not delve into this capability.

The Systems Modeling Language (SysML) is based on the Unified Modeling Language (UML) [12] but caters in both semantics and usage specifically towards systems engineering as opposed to UML's focus on software development. Many of the basic elements, interactions, and views from UML are included or extended within SysML. A key difference between the two graphical languages is a shift from UML's Class definition to SysML's Block definition, which is used to represent systems and their components. A set of views is also included in SysML to allow for constraint and requirements analysis. These provide for a parametric analysis [13] and requirements traceability capabilities, respectively.

There are four sets of viewpoints that are captured in SysML – structural, behavioral, requirements, and parametrics [14]. Structural viewpoints establish the definition of elements – the composition of systems, their properties, and organizational grouping. Behavioral viewpoints describe how these elements function, their

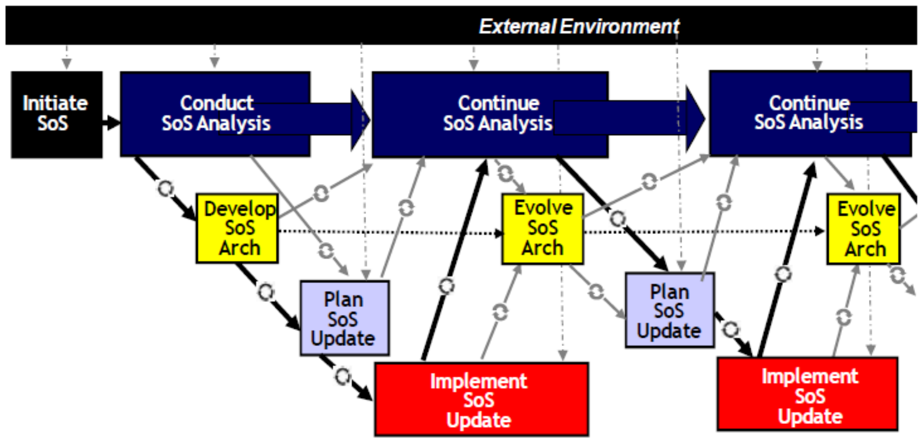


Figure 1.1. Wave model for System-of-Systems Engineering

operational states, and their interactions. The requirements viewpoint allows for a systems engineer to create, relate, trace, and analyze the formal requirements within the SoS. Finally, parametric viewpoints allow for the application of constraints on systems via logical and mathematical expressions. Each of these sets of viewpoints is used and discussed in this thesis.

1.4 Research Objectives

The goal of this research is to assess the applicability of SysML to perform key activities in SoS Engineering. To do this, the Wave model for SoSE is followed (see Figure 1.1, as outlined by Dahmann [15] and summarized in Chapter 3 of this thesis. To examine a full scope of SoS's, each phase of the SoSE Wave model will be executed for directed, acknowledged, and collaborative examples of SoS:

SoS Initiation and Early-Phase Analysis

Establish the foundational information to begin the SoSE process and analyze the SoS operations and structure in order to form a baseline for SoS evolution.

1. How does SysML define systems and their relations within a SoS?

2. How does SysML represent the behavior of constituent systems and allocate them to the systems of a SoS?

SoS Architecture Development and Evolution

Create the framework to allow for SoS evolution and develop a migration plan.

1. How are alternative configurations, system sets, and/or networks established and evaluated in SysML within the SoS scope?
2. What methods within SysML exist for examining the evolution of the SoS?

SoS Planning for Updates and Implementation

Evaluate the SoS in order to plan for the next SoS upgrade cycle and implement changes to the SoS.

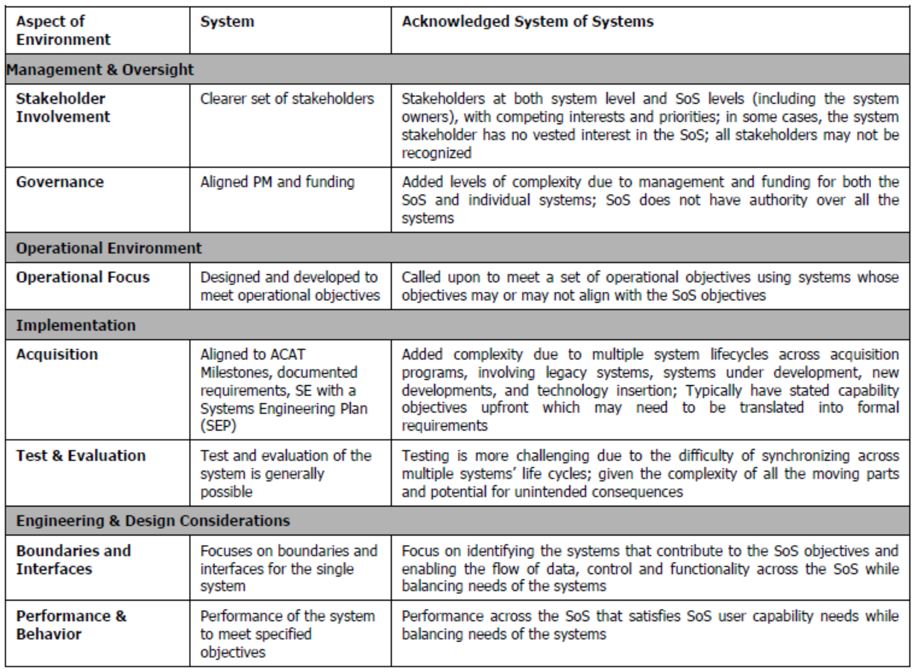
1. What analysis methods does SysML provide for the SoS?
2. How well does SysML execute these analysis methods?

The focus of this thesis is to analyze SysML for its ability to define, evaluate, and evolve a SoS model. All steps of the SoSE Wave model are carried out within SysML, both stressing the capabilities of the language and analyzing its effectiveness. Within each step, directed, acknowledged, and collaborative examples of a SoS are examined. Though the Wave model is designed for use with the acknowledged SoS type, the processes in each step are adapted for the directed and collaborative types.

Two of the Wave model steps are not carried out in SysML: Initiation and Implementation. SoS Initiation deals with problem definition, outlining what the SoS is and establishing its scope. This abstract definition is most easily done outside of SysML with a pen-and-paper approach as a conceptualization step for the modeling effort. At the end of the Wave model, the Implementation step solely deals with executing the changes examined in previous steps; no additional SysML modeling is required for this step.

The DoD *Systems Engineering Guide for Systems of Systems* identifies some of the key differences between systems and SoS. A comparison of these two fields across

Table 1.1 Comparison of Systems and Acknowledged SoS [9]



Aspect of Environment	**System**	**Acknowledged System of Systems**
Management & Oversight		
Stakeholder Involvement	Clearer set of stakeholders	Stakeholders at both system level and SoS levels (including the system owners), with competing interests and priorities; in some cases, the system stakeholder has no vested interest in the SoS; all stakeholders may not be recognized
Governance	Aligned PM and funding	Added levels of complexity due to management and funding for both the SoS and individual systems; SoS does not have authority over all the systems
Operational Environment		
Operational Focus	Designed and developed to meet operational objectives	Called upon to meet a set of operational objectives using systems whose objectives may or may not align with the SoS objectives
Implementation		
Acquisition	Aligned to ACAT Milestones, documented requirements, SE with a Systems Engineering Plan (SEP)	Added complexity due to multiple system lifecycles across acquisition programs, involving legacy systems, systems under development, new developments, and technology insertion; Typically have stated capability objectives upfront which may need to be translated into formal requirements
Test & Evaluation	Test and evaluation of the system is generally possible	Testing is more challenging due to the difficulty of synchronizing across multiple systems' life cycles; given the complexity of all the moving parts and potential for unintended consequences
Engineering & Design Considerations		
Boundaries and Interfaces	Focuses on boundaries and interfaces for the single system	Focus on identifying the systems that contribute to the SoS objectives and enabling the flow of data, control and functionality across the SoS while balancing needs of the systems
Performance & Behavior	Performance of the system to meet specified objectives	Performance across the SoS that satisfies SoS user capability needs while balancing needs of the systems

various environments is displayed in Table 1.1. Because SysML is designed for SE practices, specific modeling patterns are used to extend the language for SoSE use. The conclusions from this examination are presented in a similar table found in Chapter 6.

2. BACKGROUND

2.1 Systems-of-Systems Engineering Wave Model

As mentioned earlier, Systems-of-Systems Engineering grew from the previous use of Systems Engineering. The SE process, primarily utilized for a single system, goes through a series of steps from the conceptualization of the system through its creation, testing, and product lifecycle. Beginning with a need for a system, provided by customers and other stakeholders, requirements are generated. A conceptualization of the system is developed from these requirements, often through the use of various formalized techniques (e.g., brainstorming, QFD) [16]. Narrowing down a set of conceptual designs results in the design to be implemented. Before the system is deployed for operation, it is iteratively tested and redesigned until a final design is used for production. This entire sequence has been captured in a variety of process models: the waterfall model, the spiral model, and the Vee process model being some of the most popular.

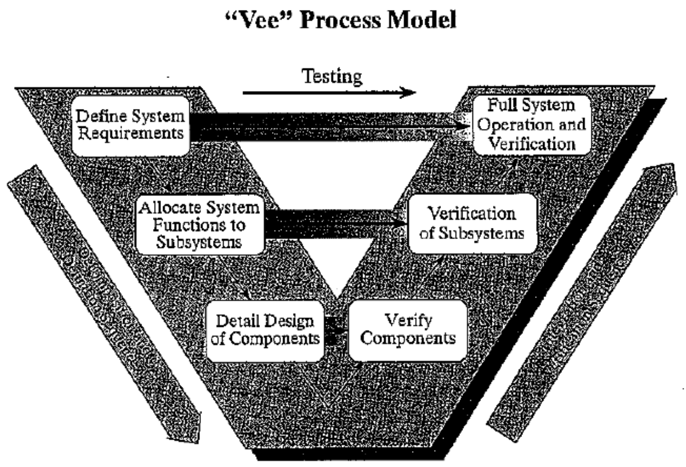

"Vee" Process Model
Testing
Define System Requirements
Full System Operation and Verification
Allocate System Functions to Subsystems
Verification of Subsystems
Detail Design of Components
Verify Components

Figure 2.1. The Vee process model for systems engineering

The Vee process model, seen in Figure 2.1, follows a decomposition from high-level requirements to the system design and then a series of test and evaluation steps that parallel the earlier steps. Each of these latter steps functions as a verification or validation of the design steps preceding it. The System-of-Systems Engineering process used in this research, known as the Wave model, has much in common with the Vee process model.

The Wave model [15] exists as a way to outline the necessary steps for engineering a SoS. Built from the "trapeze model" [9], which details several of the necessary elements required for SoSE, the Wave model linearizes this process into a set of repeated steps that occur over time. Similar to the Vee process model, the Wave model shows a process that includes analysis, development, and implementation, with feedback between these steps. However, unlike the Vee process depicted in Figure 2.1, the Wave model explicitly depicts the analysis and resulting feedback throughout the entire process. The temporal aspect of the Wave model best captures the evolutionary nature of a SoS with its repeated iterations of analysis on the SoS. Changes may come from within the engineering process or from the external environment, which has continual influence on the SoS. Finally, the fact that the Wave model is forward-moving through time allows for practitioners to directly adapt a representation of the SoS – in this case, the model – to a specific development plan.

The Wave model is broken into five major steps. First, the SoS is initiated, establishing the foundational information of the SoS – key users, objectives, and systems. After the initiation of the SoS, there exists an ongoing effort to analyze the SoS. This step evaluates the current state, or baseline, of the SoS and establishes initial plans for evolution. During this step, a technical baseline, performance measures, requirements, and planning elements are all established. The SoS architecture is then developed or evolved. The first time this step is performed, the SoS architecture is created by developing the systems and functions as well as the relationship and interactions between them. When evolving the SoS architecture, these elements are re-defined or expanded. The next step is to plan SoS updates; the SoS is evaluated and a baseline is created. A plan involving risk mitigation and testing is created in order to facilitate the evolution of the SoS from one state to the next. This plan is then implemented while SoS engineers (or related team members) facilitate and monitor the process. This results in a new SoS baseline that feeds into the SoS analysis step again, repeating the process throughout the SoS lifecycle.

This research follows each of these steps through a MBSE approach in SysML. Each step is performed with a notional SoS and evaluated. Though the Wave model was developed for use with an acknowledged SoS, the key elements of each step are adapted and applied to directed and collaborative SoS demonstration architectures as well.

2.2 Systems Modeling Language

SysML was created from the Unified Modeling Language (UML), version 2.0, with the express intent of creating an extendable, visual language with which systems engineers could work. From UML, many objects, relations, and views were adopted, while other aspects of SysML were created to capture common products during systems development. Specifically, the requirements and parametric views and all related elements are new in SysML, providing a systems engineer with methods for analyzing systems and the requirements that must be fulfilled by them [14].

The Block is the base structural element of SysML. It is commonly used to represent a system, component, or an aggregation of either element. Blocks are allowed to have several different properties including: part properties denoting the composition of a Block, value properties specifying parameters of the block, and reference properties to elements that are utilized by the Block. Relations between Blocks are displayed via *block definition diagrams* (bdd) while the properties of a single block and their interactions are displayed via *internal block diagrams* (ibd).

The behavioral aspect of a system or SoS is captured with SysML activity and action elements. These elements establish the operations that are carried out by

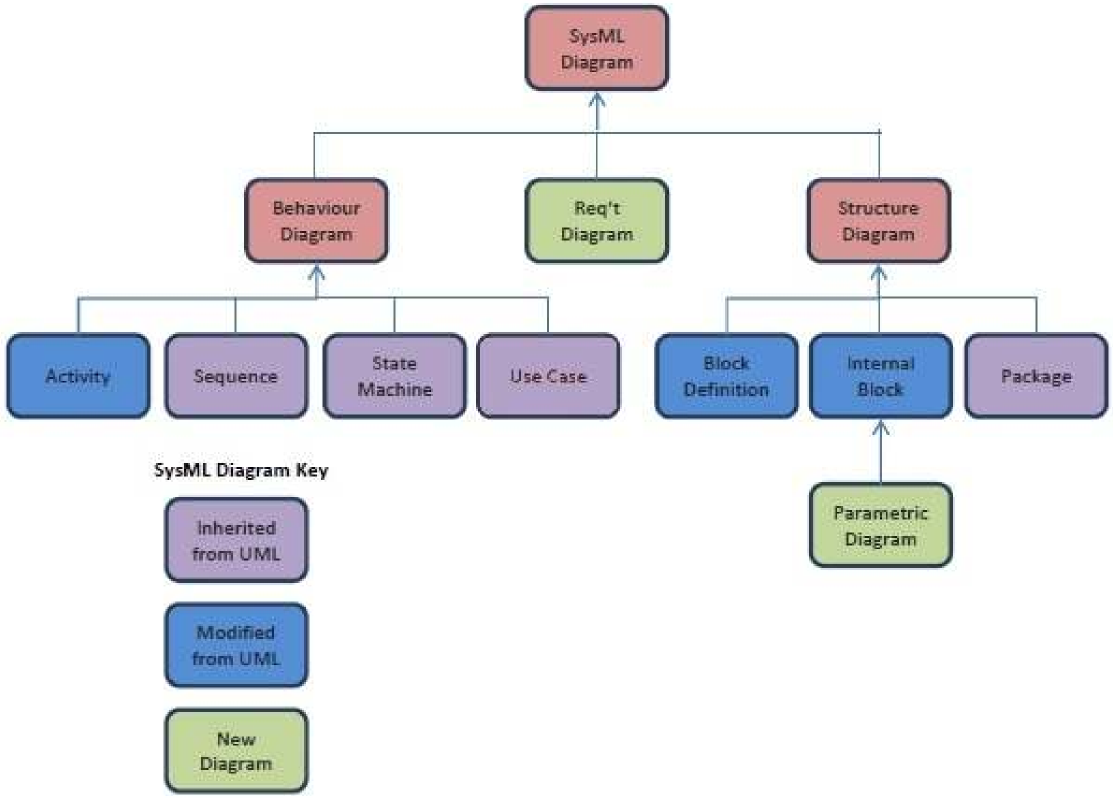


Figure 2.2. A breakdown of the various viewpoints within SysML

systems and can be related with *activity diagrams*. Activity diagrams are similar to Functional Flow Block Diagrams from Systems Engineering [17] and show the process flow within an activity. This flow can be composed of other activities with their own internal processes or actions, which are singular processes. Sequence diagrams may also be used to show logic- or time-based sequences of message exchanges between system components.

Two other aspects of SysML are discussed: requirements and parametrics. Both are additions to UML included in SysML to meet SE needs. Requirement elements and the related *requirements diagrams* allow practitioners to create requirements and tie them to model elements for verification and test planning. Analysis can be performed on these requirements to ensure that the systems represented by model elements adhere to defined constraints and that all requirements defined are met. Parametric elements allow for calculations to be performed on system properties, useful for performance analysis.

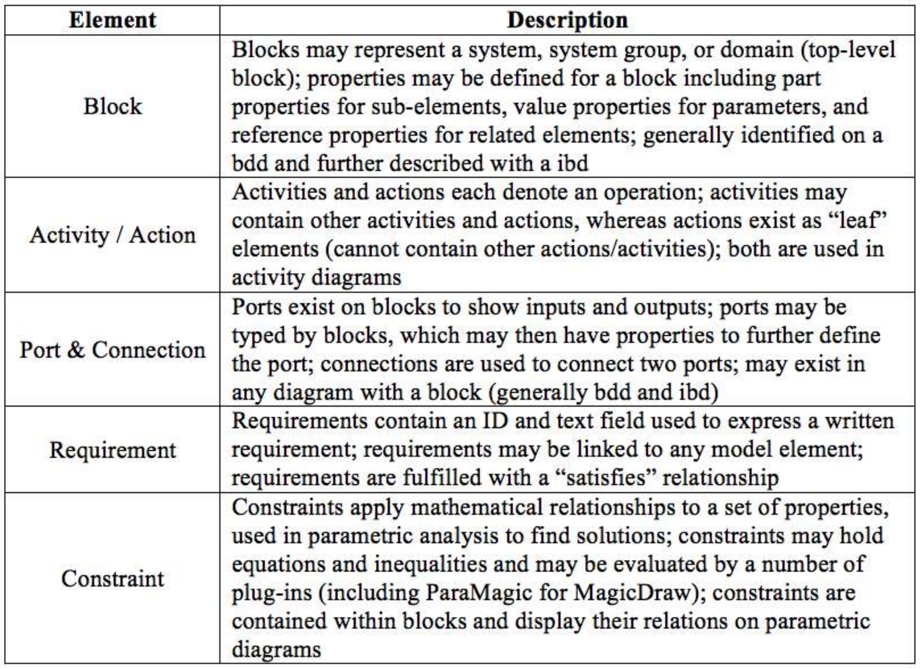
For reference, an overview of the SysML elements and diagrams used in this thesis are summarized in Tables 2.1 and 2.2.

2.3 Previous Work

Currently, SysML is mostly being utilized for Systems Engineering, as it was designed [18]. However, as stated in the Introduction, systems are becoming more complex and network-centric, exhibiting the managerial and operational independence traits required to categorize them as SoS's [3]. Efforts to utilize SysML for SoS have focused on a singular problem or on limited aspects of SysML. Though this thesis follows a single problem domain (distributed sensor management), that domain is explored in various ways to ensure that the key aspects of each SoS type are evaluated for SysML's applicability.

Lane and Bohn have written about utilizing SysML for SoSE, but keep to a strictly limited usage of SysML and focus solely on a single acknowledged SoS problem [19].

Table 2.1 Key SysML Elements



Element	Description
Block	Blocks may represent a system, system group, or domain (top-level block); properties may be defined for a block including part properties for sub-elements, value properties for parameters, and reference properties for related elements; generally identified on a bdd and further described with a ibd
Activity / Action	Activities and actions each denote an operation; activities may contain other activities and actions, whereas actions exist as "leaf" elements (cannot contain other actions/activities); both are used in activity diagrams
Port & Connection	Ports exist on blocks to show inputs and outputs; ports may be typed by blocks, which may then have properties to further define the port; connections are used to connect two ports; may exist in any diagram with a block (generally bdd and ibd)
Requirement	Requirements contain an ID and text field used to express a written requirement; requirements may be linked to any model element; requirements are fulfilled with a "satisfies" relationship
Constraint	Constraints apply mathematical relationships to a set of properties, used in parametric analysis to find solutions; constraints may hold equations and inequalities and may be evaluated by a number of plug-ins (including ParaMagic for MagicDraw); constraints are contained within blocks and display their relations on parametric diagrams

Table 2.2 Key SysML Diagrams

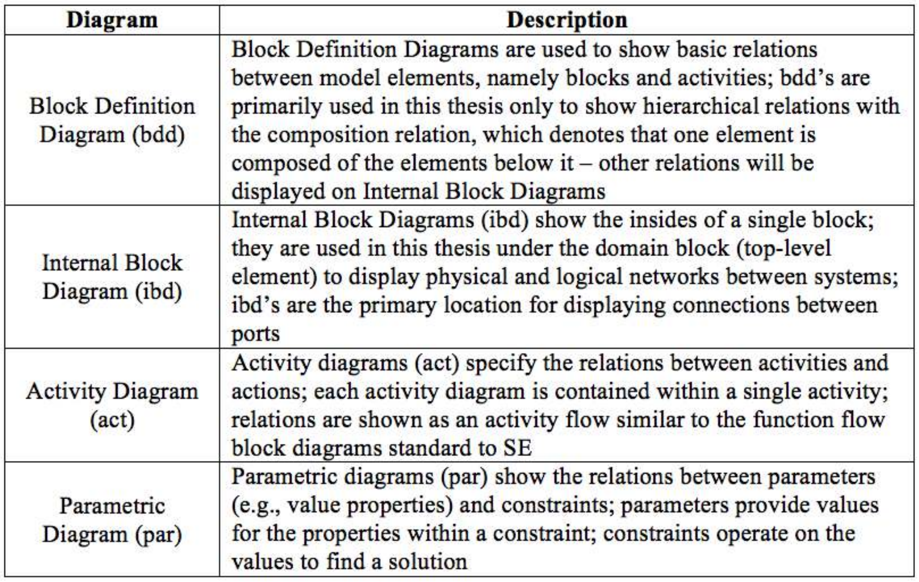


Diagram	Description
Block Definition Diagram (bdd)	Block Definition Diagrams are used to show basic relations between model elements, namely blocks and activities; bdd's are primarily used in this thesis only to show hierarchical relations with the composition relation, which denotes that one element is composed of the elements below it – other relations will be displayed on Internal Block Diagrams
Internal Block Diagram (ibd)	Internal Block Diagrams (ibd) show the insides of a single block; they are used in this thesis under the domain block (top-level element) to display physical and logical networks between systems; ibd's are the primary location for displaying connections between ports
Activity Diagram (act)	Activity diagrams (act) specify the relations between activities and actions; each activity diagram is contained within a single activity; relations are shown as an activity flow similar to the function flow block diagrams standard to SE
Parametric Diagram (par)	Parametric diagrams (par) show the relations between parameters (e.g., value properties) and constraints; parameters provide values for the properties within a constraint; constraints operate on the values to find a solution

They conclude that SysML cannot meet all of the needs of SoSE but is a useful tool for organizing and integrating information from across a SoS design space. However, their use of SysML is limited and restricts its capability. An improvement upon the methods of Lane and Bohn is to move away from the use of sequence diagrams, which limit the adaptability of a model to meet a changing system by tightly coupling the behavioral and structural definition of interactions between systems. This restricted definition, along with Lane and Bohn's assertion that SysML cannot dynamically execute models, result in their conclusion that SysML cannot perform all SoSE tasks.

The research proposed in this thesis expands on many of the methods of Lane and Bohn. First, a different approach to defining system interactions is proposed, which allows for more flexibility in definition and application of behaviors among structural elements (systems and components). Secondly, by utilizing software plugins to extract information from a model and interpret it, an execution of the model is capable. Other areas of research that have dealt with SysML and SoS focus on how to perform a single task within SysML, for example requirements engineering [20] and cost modeling [22]. These two areas are briefly discussed but are not key to the scope of this thesis as they both function as a parallel and independent query of a SysML model.

3. SOS INITIATION & EARLY-PHASE ANALYSIS

The goal of SoS Initiation is to establish the foundational information with which to begin the SoSE process. Initiation generates the objectives, key users, and systems of the SoS. This step also provides the scope of the SoS that will be examined in this thesis – a regional distributed sensor architecture. The other step discussed in this chapter is the recurring SoS Analysis, which establishes the "as-is" baseline architecture for the SoS. As SoS Analysis is an ongoing process during each iteration of the Wave model, this chapter focuses on the early-phase analysis as the SoS is first defined. A Concept of Operations (CONOPS) is created, keys systems are modeled, and the structure of the SoS is defined.

To examine the applicability of SysML to SoS, an example problem domain is examined to explore several facets of SoSE. The problem deals with the management and operation of multiple distributed sensors by multiple command nodes. Key to these elements are how the functionality of controlling the sensors is allocated among the constituent systems and how the systems are networked together to allow for information exchange while operating independently. This problem domain will be examined in the context of directed, acknowledged, and collaborative SoS's.

In each SoS type, individual radar sensors independently operate to detect and track various targets. The overall goal of each SoS is to generate high-accuracy tracks on all targets in the field. Radars and their Command & Control (C2) nodes communicate in order to improve their performance and the overall performance of the SoS. In the cases of the directed and acknowledged SoS's, higher level objectives require the radars and C2 nodes – organized into control groups – to alter their behavior. In the case of the collaborative SoS, the higher level objective is realized through rules placed on the interactions of control groups.

3.1 SoS Initiation

To initiate the SoS, the problem domain must first be defined. Each SoS example (directed, acknowledged, and collaborative) incorporates many of the same systems and activities, but differ in how the systems operate and interact. Specifically, each SoS example focuses on the interactions between radar sensor platforms – a physical entity housing one or more radar sensors and the necessary subsystems to operate them (note: these subsystems are not within the scope of the model). These radars are capable of independently measuring and tracking targets and interact with Command & Control (C2) nodes in order to improve the radars' performance. Each group of radar platforms managed by a C2 node are organized into control groups. With the interaction of control groups, the behavior of radar systems may change based on the higher-level objects of the SoS. Though this is a basic example, it retains the key features of a SoS and allows for these features to be explored within SysML.

First and foremost, the operational and managerial independence of the systems are established. Each system is defined with the ability to operate by itself and has some level of governance over its own actions. Each radar by itself retains the ability to scan an area and track an object if one is found. However, when it is combined with other radars and C2 nodes, it gains the added knowledge of the other sensors and can seek out objects to track, displaying emergent properties. Each group of radar sensors and their interacting C2 node is defined as a control group. These control groups each maintain their own managerial independence, though their behavior may be impacted by high-level objectives. Systems are all geographically distributed, networked, and may be defined heterogeneously. Finally, the Wave model approach focuses on the evolutionary nature of this SoS as this feature will be examined in particular for its relation to SysML in the next chapter. The latter parts of this chapter will focus on how SysML may be used to capture the definition of these systems and their interactions.

Table 3.1 ROPE-Scope Table for Distributed Sensor Problem

Level	Resources	Operations	Policy	Economics
ε	Global assets	Global command	Global goals and operations policies	Global resource management and acquisition
γ	Regional assets	Regional command	Regional goals and operations policies	Regional resource management trades, systems acquisition
β	Control groups	Command, control, measurement and track fusion	Interface requirement policies	Economics of multiple sensors and C2 nodes
α	Sensors, platforms, communication nodes, C2 nodes	Tasking, measurement generation	System operations procedures mission profiles	Economics of acquiring and operating single resources

Common in the SoS field is the creation of a ROPE-Scope table to outline the resources, operations, policies, and economics of the SoS at various levels of abstraction [21]. The ROPE-Scope table for the distributed sensor SoS is shown in Table 3.1. Each level of abstraction builds upon the level below it (e.g., sensors and C2 nodes are aggregated into control groups). The scope of this research will focus on the α, β, and γ levels of abstraction for simplicity. Furthermore, the resource and operations aspects of the SoS are the focus of much of the SoS engineering in this thesis. Policies are discussed briefly due to their relationship to requirements; this discussion is brief due to the ease of implementing SoS requirements in SysML [20]. The economics of the architecture definition and integration effort for software-intensive, net-centric SoS's are covered in the work on the Constructive SoS Integration Model (COSOSIMO) [22].

3.2 Resource Definition

Proceeding directly from the ROPE-Scope table (see Table 3.1), the resources used in each SoS are first defined. In SysML, this type of hierarchical system definition is done within a block definition diagram (bdd). Various blocks are created for various systems at different levels of abstraction and relations may be drawn between them to establish the composition of one level with the ones below it.

Figure 3.1 establishes the hierarchy of systems. As is the case in the ROPE-Scope table, the SoS architecture is composed of control groups, which have sensors and C2 nodes. This top level block exists to encapsulate the different systems in order to allow them to interact. It follows the ROPE-Scope table closely, but deviates in certain aspects for modeling purposes. For example, communication (comm.) nodes are established directly below the SoS architecture block; this is due to the fact that a comm. node can exist and interact at any level in the SoS. Similarly, the targets to be tracked are created under the SoS architecture block, despite being external

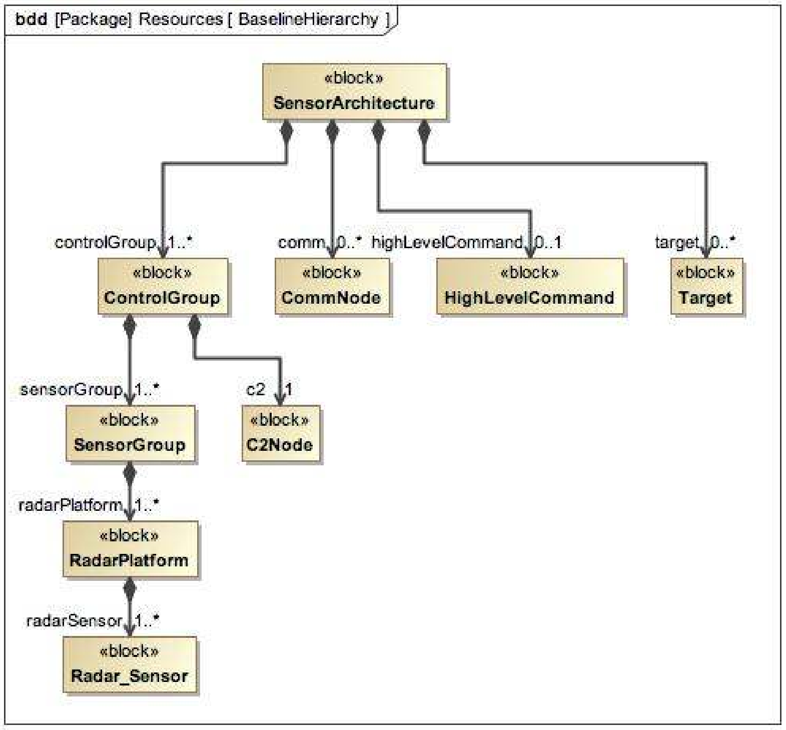


Figure 3.1. Block definition diagram showing the baseline hierarchy of systems within the SoS

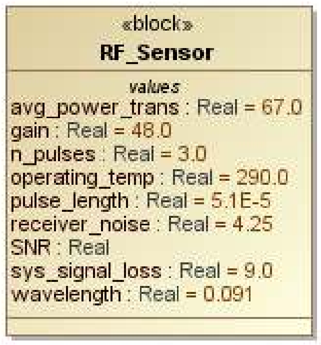

«block»
RF_Sensor
values
avg_power_trans : Real = 67.0
gain : Real = 48.0
n_pulses : Real = 3.0
operating_temp : Real = 290.0
pulse_length : Real = 5.1E-5
receiver_noise : Real = 4.25
SNR : Real
sys_signal_loss : Real = 9.0
wavelength : Real = 0.091

Figure 3.2. Value properties and default values defined for a Radar Sensor block

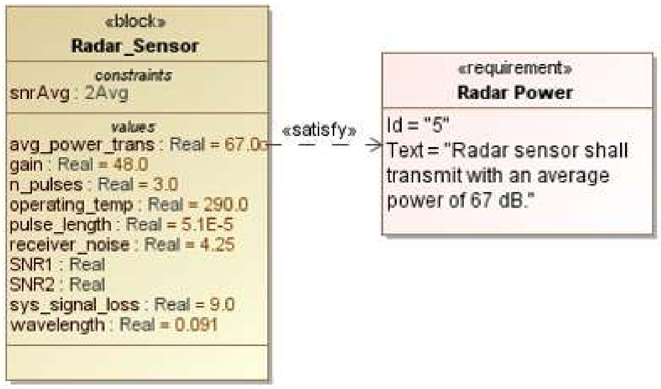

«block»
Radar_Sensor
constraints
snrAvg : 2Avg
values
avg_power_trans : Real = 67.0
gain : Real = 48.0
n_pulses : Real = 3.0
operating_temp : Real = 290.0
pulse_length : Real = 5.1E-5
receiver_noise : Real = 4.25
SNR1 : Real
SNR2 : Real
sys_signal_loss : Real = 9.0
wavelength : Real = 0.091
«satisfy»
«requirement»
Radar Power
Id = "5"
Text = "Radar sensor shall transmit with an average power of 67 dB."

Figure 3.3. Example requirement on the avg. transmitted power of a Radar Sensor block, satisfied by its value property

elements to the SoS. This is done to later allow calculations between the radar sensor and target blocks.

Blocks can be imbued with various properties to further define their attributes. For the applications of this research, the value property is most important. Value properties allow a variable to be defined with a specific numerical value. In this way, attributes of a system (e.g., location, radar aperture, measure rate, comm throughput) can be parameterized for analysis.

A single default value can be established for each value property on a block. However, this default value applies to every instance of the block. To allow for heterogeneity among systems of the same block type, specific configurations of a block are created via generalization relations between the "general" block and its "specific" instantiation. Any properties, ports, and relations that the general block contains are inherited by the specific block. The set of value properties for a radar sensor block are seen in Figure 3.2 and are later used in calculations for signal-to-noise ratio on measurements.

For traceability of system requirements, value properties are linked to Requirement blocks through the satisfies relation. Based on when a model is being created within the SoS SE process, a set of system requirements may already exist or a set may be created from the model. If a set already exists, they should be imported to generate a set of requirement blocks. The imported requirements can then be attached to model elements via a *satisfies* relationship in order to later perform requirement analysis on the current set. If a set does not exist, requirements should be created to satisfy system and SoS properties. An example requirement and its *satisfies* relationship are shown in Figure 3.3. The use of requirements and their analysis capabilities will be discussed further in Chapters 4 and 5.

3.3 Develop a Concept of Operations

The next major step in analyzing the SoS is to develop a Concept of Operations (CONOPS). This step defines the details of the operations outlined in the ROPE-Scope table. This is accomplished in SysML via the use of activities and their respective diagrams.

A *Block Definition Diagram* (bdd) of activities is generated to lay out the hierarchy of operations. This can be seen in Figure 3.4 and details a functional decomposition. Not all of these cases may be used in every scenario, depending on the configuration or situation, but they are meant to span the full space of operations. The set defined in this research only deals with nominal operation and will be used with each SoS type.

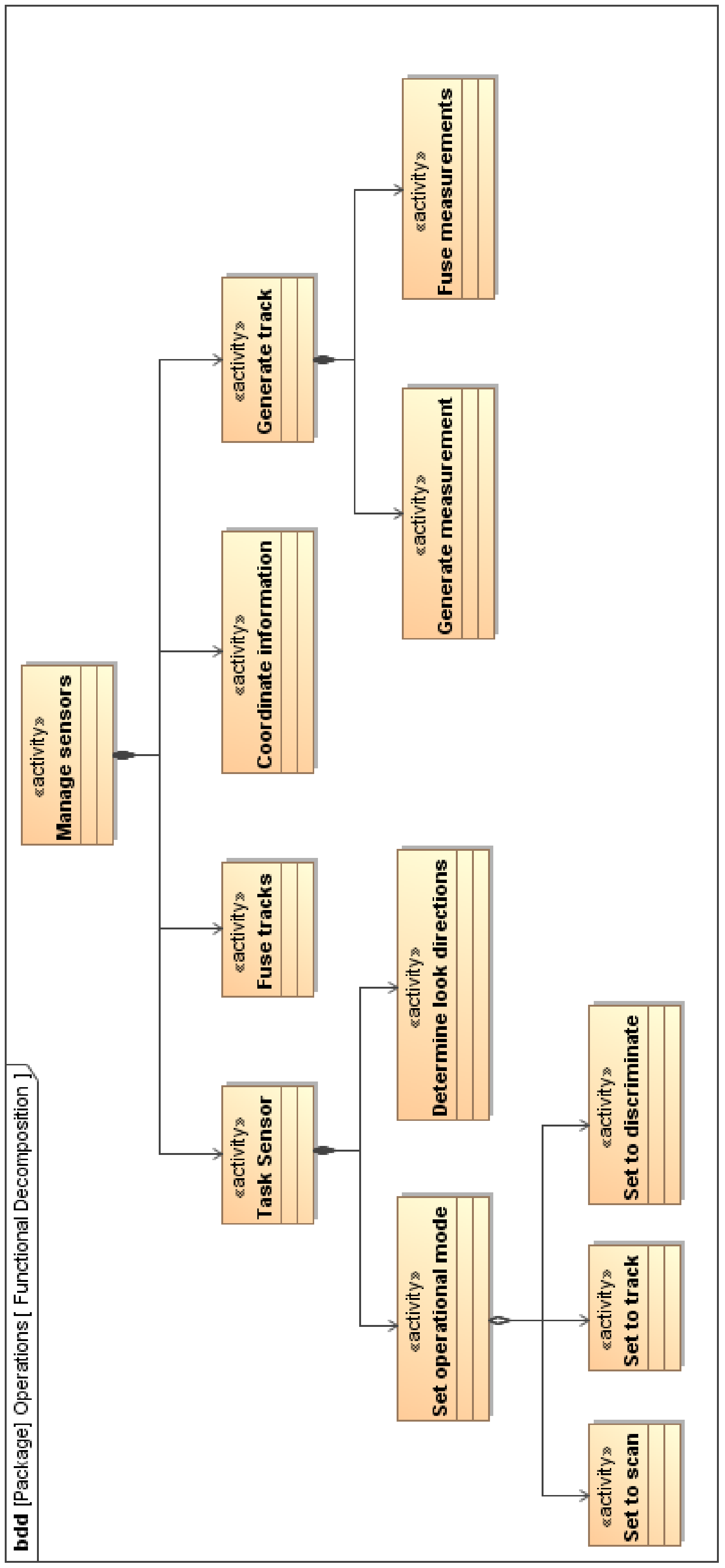

bdd [Package] Operations [Functional Decomposition]
«activity» Manage sensors
«activity» Task Sensor
«activity» Fuse tracks
«activity» Coordinate information
«activity» Generate track
«activity» Set operational mode
«activity» Determine look directions
«activity» Generate measurement
«activity» Fuse measurements
«activity» Set to scan
«activity» Set to track
«activity» Set to discriminate

Figure 3.4. Functional decomposition developed for CONOPS

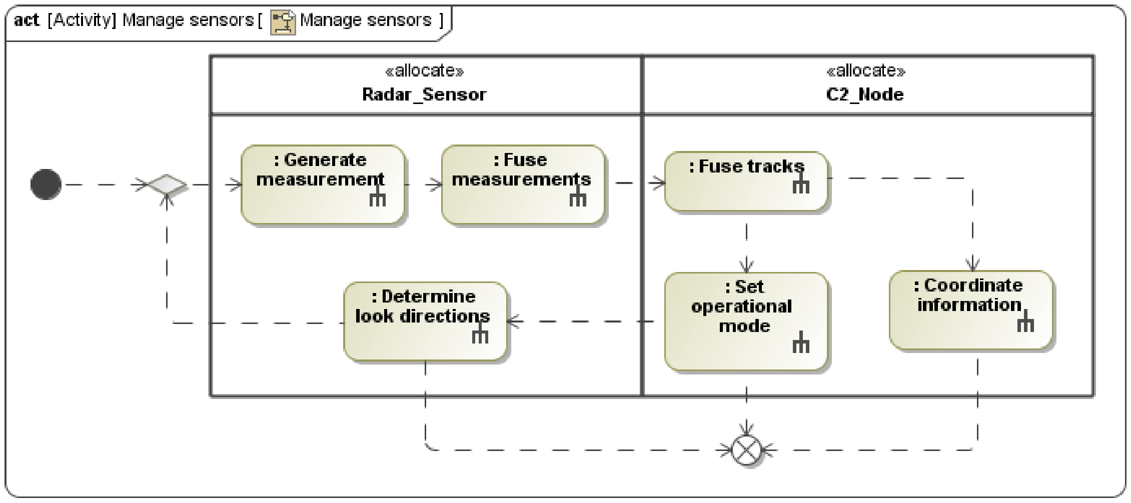


Figure 3.5. Example activity flow: the allocation and flow of key activities

The hierarchical decomposition of activities shows that the overall function of the SoS is the regional command of assets. This top-level activity includes tasking sensors, generating tracks, fusing tracks, and coordinating information throughout the various systems. In order to task a sensor, an operational mode must be selected and look directions must be determined. Tracks are generated by generating measurements and fusing them into tracks.

At this point, a certain flow of activities can be seen as necessary. Measurements have to be generated first and fused together to make tracks. Tracks can be fused together to inform sensor tasking and much of this information can be coordinated between sensors and/or command & control nodes. How this activity flow occurs and how the activities are allocated to systems are defined through *Activity Diagrams*.

Figure 3.5 shows the activity diagram for the top-level sensor management activity. Within this diagram, the flow of activities within the scenario is established as previously described. The directed, dashed lines between activities are control flow connections and convey the procession of activities.

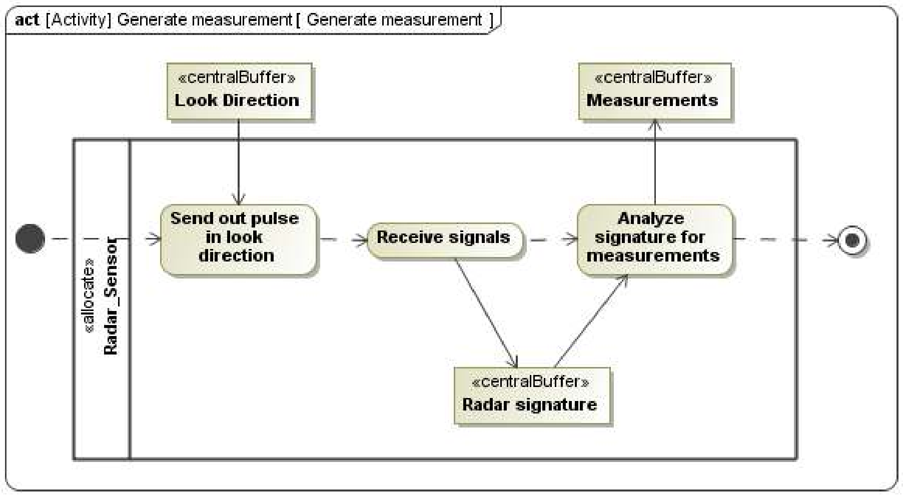

act [Activity] Generate measurement [Generate measurement]
«centralBuffer» Look Direction
«centralBuffer» Measurements
«allocate» Radar_Sensor
Send out pulse in look direction
Receive signals
Analyze signature for measurements
«centralBuffer» Radar signature

Figure 3.6. Example activity flow: actions required for a radar sensor to take a measurement

In SysML there exists a methodology for allocation of elements to one another via a stereotyped dependency relationship. Therefore, in order to allocate capabilities to systems, an allocation relationship is established between activities and the systems performing those relationships. This relationship is also created with the use of allocation swimlanes, visible in Figure 3.5, where the activities within the swimlane of each block are allocated to that block.

How these activities are allocated to the various system blocks is a key factor in determining the type of SoS being modeled. High functionality for low-level resources is more likely to result in a collaborative or virtual SoS, with little to no centralized control over systems. On the other hand, allocating more functionality at a high-level system yields a more centralized, directed SoS. Section 3.4 discusses the different functional allocations and configurations for each SoS type.

Along with control flows, object flows (solid directed lines) show the flow of information through activities. The Radar Measurement activity, whose activity diagram is seen in Figure 3.6, dictates that a radar will start the measurement activity by sending out a pulse in a specific look direction. To complete this action, it receives a Look Direction object – an abstraction of the information that signals the radar to point in a specific direction. Such an abstraction can be further defined by specializing or defining its composition (both are relations in SysML). After receiving signals, a radar signature is generated and then analyzed to create measurements. An activity is analogous to a function block within a set of code; it has a self-contained definition with inputs and outputs to allow it to string together with other activities.

3.4 Application to SoS Types

Having defined the basic structural and functional building blocks, it is possible to create specific SoS architectures. Different configurations of the base architecture are generated for each of the three SoS types used in this research: directed, acknowledged, and collaborative. The resources and operations defined earlier in this chapter are assembled and characterized for each SoS type differently to emphasize the properties of that SoS.

In order to allow the properties and behavior of each block to be individually designated, allowing for variants on each of the systems defined, "redefinition properties" are utilized. A redefinition property allows for a property of a block to override an inherited property (required due to SysML's strict definition of inheritance). Due to the loose definition of the base architecture (non-specific multiplicities and an all-inclusive block hierarchy), part properties are redefined to designate the exact configuration of systems involved in a specific SoS. Value properties are redefined in order to set the default value of a property within a configuration. To make the views in this section more readable, the redefinition context (a tag containing the name of the redefined property) is hidden; all relations will look the same as standard relations.

3.4.1 Directed SoS

For the directed SoS, two control groups are specified, each with two single-sensor radar platforms and a C2 node. One control group is built upon radar sensors built for

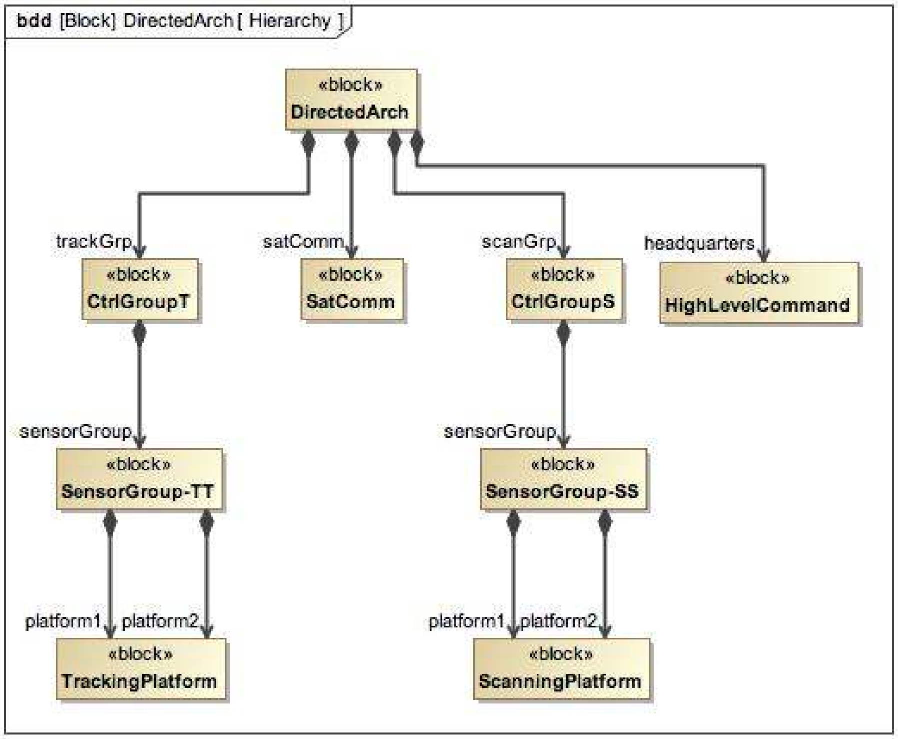


Figure 3.7. Hierarchy of resources for the directed SoS architecture

searching and detection while the other focuses on tracking detected targets. Because the directed SoS type specifies a central managing entity that can directly influence systems, an addition system is included to represent this high-level command. These resources are first uniquely defined and then related to the operations allocated to them.

To establish the hierarchy of systems for each SoS type, unique blocks are developed for each architecture. By creating these blocks separately for control groups, sensor groups, C2 nodes, and different sensor types, activities may be allocated independently to each block (i.e., not all C2 nodes are forced to operate identically). These unique blocks each inherit from their pre-defined block types to allow common properties, ports, and requirements to apply. This inheritance is done through the use of the generalization relationship between the two blocks. Figure 3.7 shows the newly defined block definitions for the directed SoS.

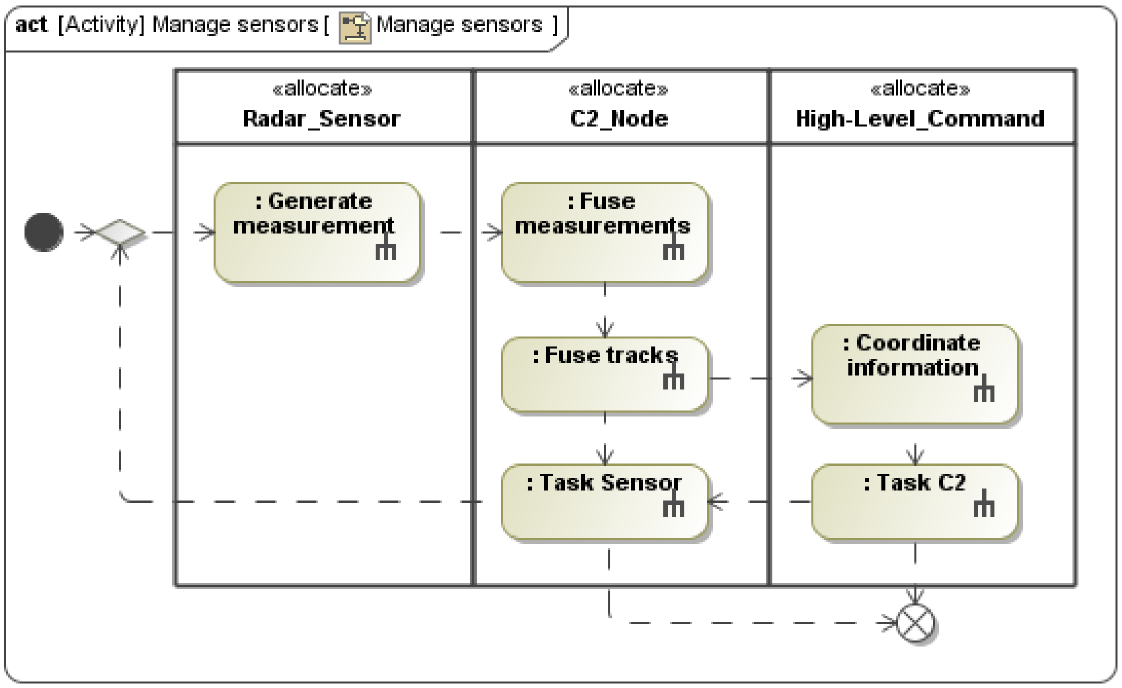


Figure 3.8. Sensor management for the directed SoS

The high-level command block is created directly under the top-level SoS block to give it the ability to interact at the top-most level with other systems. Despite the fact that all systems in this SoS are managed by a centralized manager, this block is not the aggregate of the systems below it and thus exists separately – not at the top of the tree. The high-level command is an entity within the abstract centralized management that is responsible for ensuring that SoS objectives are met.

The allocation of activities for the directed SoS relies on the control being at a higher, more centralized levels. This is shown through Figure 3.8, showing the allocation of key activities. Sensors provide raw measurements to C2 nodes, which fuse that data to generate tracks and, if applicable, fuse those tracks. C2 nodes still provide tasking to their sensors, but the coordination of information between C2 nodes is applied to the high-level command. Likewise, the high-level command is able to task the C2 nodes to influence their sensor tasking.

3.4.2 Acknowledged SoS

The acknowledged SoS architecture does not feature a centralized commanding node and therefore the high-level command block is no longer included. In this SoS, control groups manage themselves and interact to meet high-level objectives. Similar to the directed SoS, two control groups are specified, one focusing on search and detection and the other focused on tracking.

The hierarchy of systems, shown in Figure 3.9, is distinguished by the absence of the high-level command block. Furthermore, the allocation of activities in this SoS is identical to that presented earlier in this chapter (see Figure 3.5). Sensors are responsible for their own measurement fusion while C2 nodes fuse tracks, share information to other control groups, and provide tasking for sensors – outside of the sensors' capabilities to search and/or track on their own. The key difference in the architectures relies on the direct exchange of information between systems as opposed to relying on a centralized commanding node to gather data and analyze it.

3.4.3 Collaborative SoS

The collaborative SoS architecture is distinguished by the voluntary collaboration of systems; thus, each radar is grouped with its own C2 node. These control groups interact through a communication node that acts as the key relay of information between groups.

Because of the nature of collaborative SoSs being indirectly managed, the collaborative SoS model places less of an emphasis on the radar system operations and more emphasis on their interactions – namely what information is shared between systems. The goal of modeling a collaborative SoS is to analyze and establish the rules of interaction between systems, so the focus shifts from the radar systems to C2 nodes and communication nodes that manage their interactions. Each radar is capable of its own tasking and measurement and track generation. C2 nodes are then responsible for coordinating information and fusing tracks from the data they

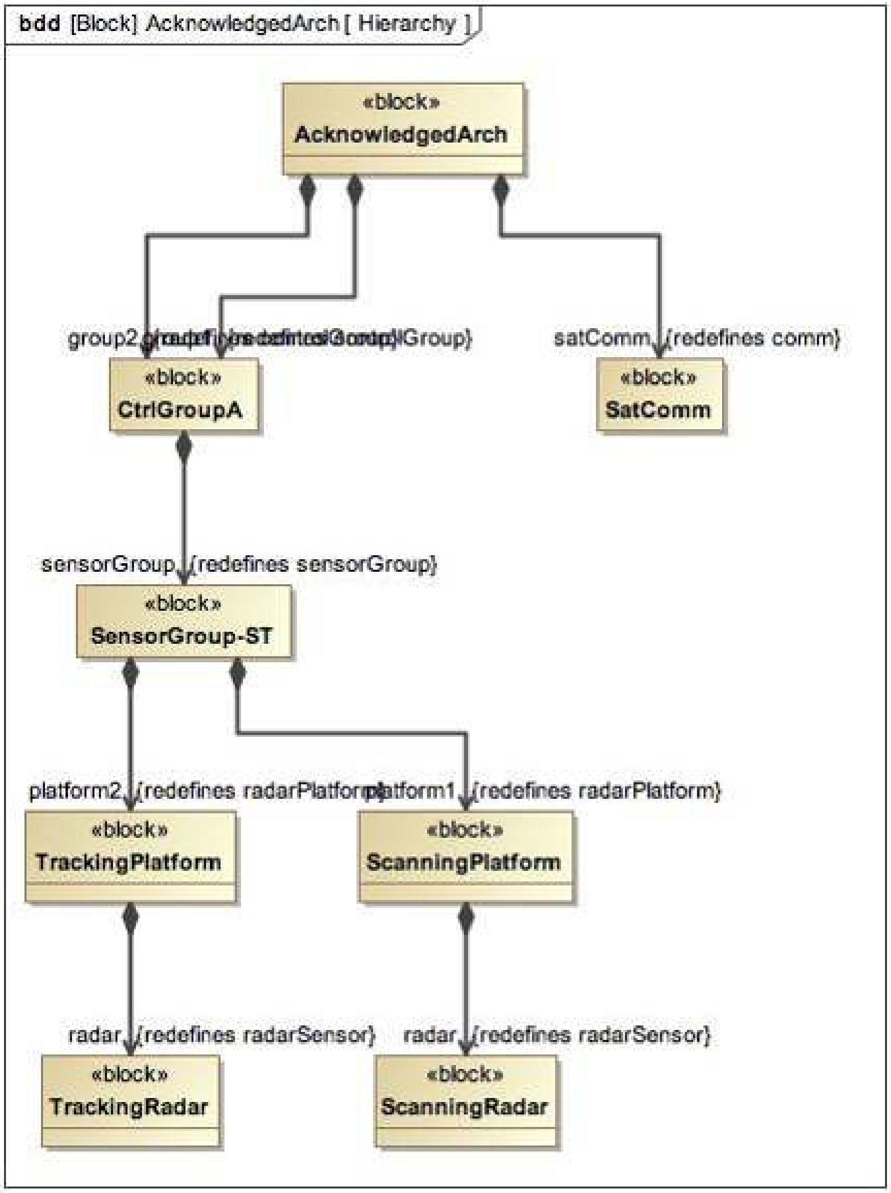

bdd [Block] AcknowledgedArch [Hierarchy]
«block»
AcknowledgedArch
group2 [illegible] Group}
satComm {redefines comm}
«block»
CtrlGroupA
«block»
SatComm
sensorGroup {redefines sensorGroup}
«block»
SensorGroup-ST
platform2 {redefines radarPlatform}
platform1 {redefines radarPlatform}
«block»
TrackingPlatform
«block»
ScanningPlatform
radar {redefines radarSensor}
radar {redefines radarSensor}
«block»
TrackingRadar
«block»
ScanningRadar

Figure 3.9. Hierarchy of resources for the acknowledged SoS architecture

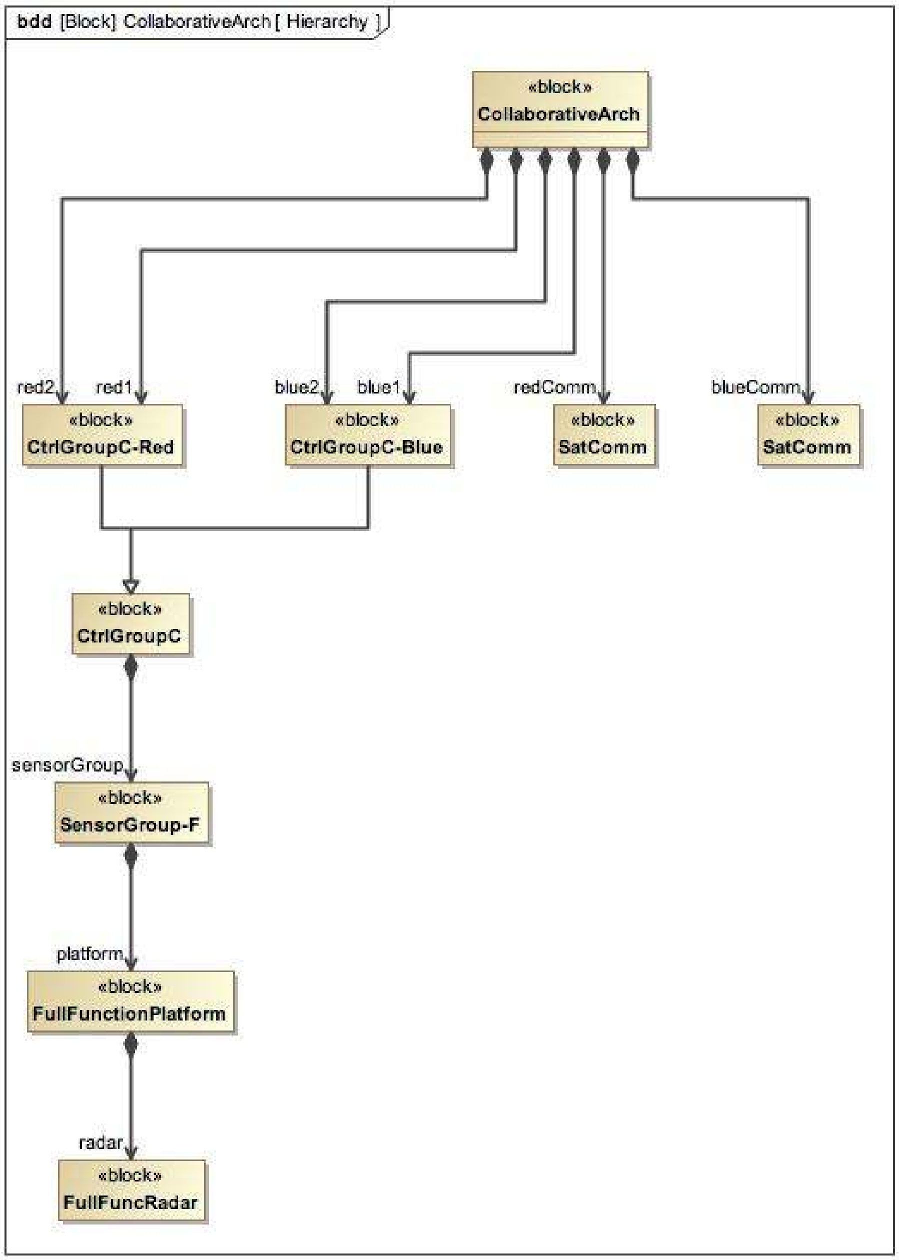


Figure 3.10. Hierarchy of resources for the collaborative SoS architecture

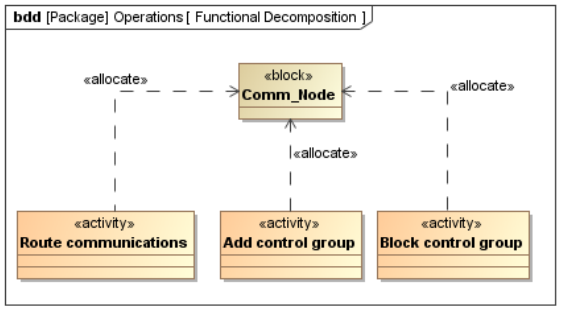


Figure 3.11. Allocation of activities to the communication node

receive. The communication node receives the added capability of deciding to and from which systems information is routed, enacting rules on the interactions between control groups. This could represent either a third-party control of information flow or logic internal to the routing processes of the communication node.

In this architecture, the communication node has the added task of determining what information to share among the control groups and which control groups to will receive information from the others. To show this, new activities are created for communication routing and adding or blocking control groups. These activities function based on rules set on how systems must interact (described in more detail in Chapter 5).

3.5 Evaluation

3.5.1 How does SysML define systems and their features within a SoS?

SysML allows for the creation of block elements to represent systems, their components, and the top-level SoS. These are hierarchically related through a series of composition relations. This process allows for the ability to define any number of levels of abstraction. A strength of SysML is its ability to capture systems and components at different levels, provide detail at those levels, and be able to view the system at those levels. Furthermore, these levels of abstraction can be easily extended upwards or downwards; detail can be provided to existing elements and higher level elements can be defined above those without requiring any change to the existing model. Being able to specify these differing levels of abstraction in a related, consistent fashion is key to SoS engineering due to the varying levels of scope.

By defining a baseline set of structural and operational elements, the common features of a SoS can be defined and inherited by different architectures with ease. However, to allow for heterogeneity between systems, it is useful to create unique blocks for each system and have them inherit properties from the baseline set of structural elements. The operational elements can then be allocated to these unique blocks independently. This adds another step in the initial definition of any SoS type and adds complexity to the model. However, once in place and used correctly – applying like features to the inherited block and unique features to the unique block – this process allows for much more flexibility in modeling different systems.

For each SoS type, the process of defining the systems and their features in SysML remains the same. The definition of systems, independently, is what SysML is designed for, and the processes developed for SE can be applied to this step in the SoS process. System definition is an aspect that shows a true strength of SysM; the process does not take a long time, accomplishes the task of establishing the base properties of a system, and initializes the model for more detailed analysis. This system definition capability is expanded upon through this research to incorporate the systems into a SoS. Basic requirements analysis can be done on each system independently; interface requirements will be discussed in Chapter 4.

Beyond defining the properties of a system within SysML, there exist plug-ins (to MagicDraw, the SysML tool used in this research) to link model elements to external software/documents that can provide more detailed information. Popular applications of these plug-ins include connecting blocks representing mechanical features to CAD parts [11] or supplying detailed orbital trajectory information through STK [10]. Furthermore, references documents, such as requirements documents, can be hyperlinked to provide further information. It is not the role of SysML to replace any of these tools/documents entirely but to be used as a tool to incorporate them into the SE or SoSE process. To use CAD as an example, instead of exporting several part drawings into a system design document, the model-centric parallel is to link each block to its CAD model directly.

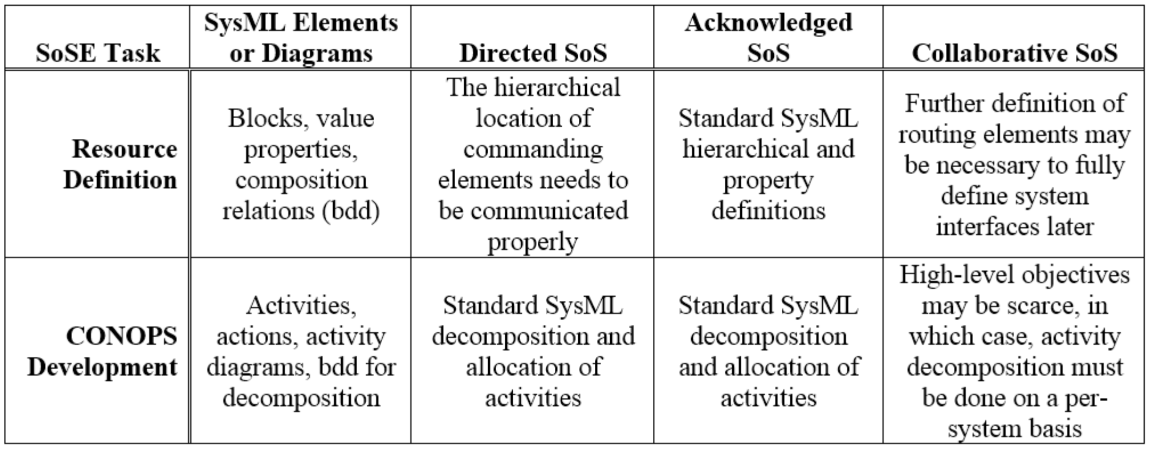
3.5.2 How does SysML define the operations and allocate them to the systems of a SoS?

The operational aspect of SysML is succinct, drawing direct inspiration from similar flowcharts that have been used in SE. A high-level CONOPS is created via an activity hierarchy, displaying a basic functional decomposition. By further defining each activity with an activity diagram, a highly detailed operational concept can be built within SysML for the SoS.

The allocation of these activities to different systems is easily done with the use of allocation lanes on activity diagrams or directly with the allocation relationship between an activity and a block or its ports. This allocation is independent of level of abstraction, allowing any combination of activities and blocks. The independence of the allocation relationship allows for vastly different levels of centralization in different SoS models drawing off the same hierarchy of activities; a feature that is useful in capturing different SoS types.

The method for defining and allocating activities remains the same for each SoS type. This is a standard SE process that transfers perfectly to SoSE due to its orthogonality. The difficulty in creating these views lies more in determining what activities must be defined than in creating them in a model. Without SysML, activity views with or without swimlanes would be created using a different graphical tool; however, SysML allows these graphical elements to be linked to other objects, maintaining a singe source of truth in the model. Though defining and allocating operations may take the same time with or without SysML, SysML allows these activities to be

Table 3.2 Overview of SoS Initiation & Analysis tasks in SysML



SoSE Task	SysML Elements or Diagrams	Directed SoS	Acknowledged SoS	Collaborative SoS
Resource Definition	Blocks, value properties, composition relations (bdd)	The hierarchical location of commanding elements needs to be communicated properly	Standard SysML hierarchical and property definitions	Further definition of routing elements may be necessary to fully define system interfaces later
CONOPS Development	Activities, actions, activity diagrams, bdd for decomposition	Standard SysML decomposition and allocation of activities	Standard SysML decomposition and allocation of activities	High-level objectives may be scarce, in which case, activity decomposition must be done on a per-system basis

used and analyzed by other parts of the model, a benefit that would not exist in a document-centric format.

4. SOS ARCHITECTURE DEVELOPMENT AND EVOLUTION

The next step in the Wave model is SoS Architecture Development and Evolution. The focus of this chapter is on creating the framework for addressing how systems interact within a configuration and later evolving these configurations. Networks are created to model system interactions while a focus is placed on allowing these networks to be re-configurable to allow for SoS evolution. Major tasks to be addressed are establishing how the systems interact, creating the networks required for those interactions, and formulating methods for altering the SoS model to demonstrate evolution.

4.1 System Interactions and Networks

An architecture is defined beyond its composition by establishing how its constituent systems interact. Two different network sets are defined, following a cyber-physical approach: the logical network and the physical network [23]. The cyberphysical approach is an abstraction of protocol layers used in networking, which show the various layers that control the conveyance of information over a network – a multitude of layers is not necessary to demonstrate the applicability of the modeling patterns used in this research. The logical network describes the exchange of information between systems – what information is transferred between systems. The physical network shows the connectivity of systems – over which physical paths the systems communicate. For example, where a logical network connection may specify that a radar platform sends data to the C2 node in its control group, the related physical network connection would show a RF link between the two. The combination of these two links show that the radar sends data over a RF connection to the C2 node, where both the action and the medium may have differing properties (e.g., data rate vs bandwidth).

Keeping these two networks separate allows for changes to either network without necessarily disrupting the other. For example, if the RF link in the previous example were replaced with a hardline connection (e.g., fiber-optic cable), the logical network would require no changes. This is in contrast to the SysML practice of using item flows to denote the information flow over a physical connection. In the case of using item flows, the physical network and logical network become directly coupled; any change to one necessitates a change to the other within the model. Though this may not be much of a hassle for a single connection, large-scale physical network changes may be common in certain SoS's –cellular data flowing through the Internet being a prime example where wifi or cellular towers may function as a medium while the functional connection remains constant.

These networks are created in SysML through the use of internal block diagrams or ibds. Each ibd is created in the context of the a single SysML block, allowing the use of all owned and inherited blocks and their properties that exist within the composition hierarchy below the context. Blocks are established with ports that detail the information that can flow in or out of the system or the physical connections that the system can support. Flowports may be uni- or bi-directional.

Figure 4.1 and 4.2 show the logical and physical network diagrams for the acknowledged SoS architecture. From the logical diagram, we can see that the C2 nodes in either control group send commands and receive data from each sensor in their group. The C2 nodes then send and receive data from each other. Based on the functional allocation and the definition of these lines, the data sent between the sensors and their C2 nodes is track information (data outlining the physical properties of a target being tracked by that sensor). The C2 nodes can communicate these fused tracks to each other to allow for better command and control of their constituent sensors. The physical network diagram shows that all of this communication occurs via RF through a singular communications node.

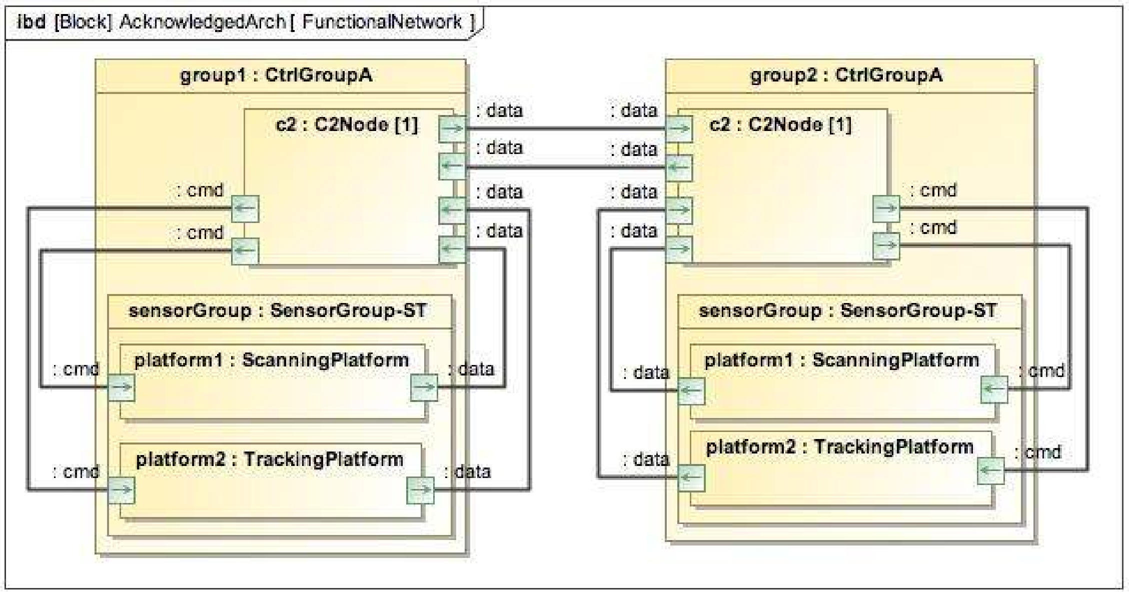

ibd [Block] AcknowledgedArch [FunctionalNetwork]
group1 : CtrlGroupA
c2 : C2Node [1]
: data
: data
: data
: data
: cmd
: cmd
sensorGroup : SensorGroup-ST
platform1 : ScanningPlatform
platform2 : TrackingPlatform
: cmd
: data
: cmd
: data
group2 : CtrlGroupA
c2 : C2Node [1]
: data
: data
: data
: data
: cmd
: cmd
sensorGroup : SensorGroup-ST
platform1 : ScanningPlatform
platform2 : TrackingPlatform
: data
: cmd
: data
: cmd

Figure 4.1. Logical network diagram for the acknowledged SoS architecture

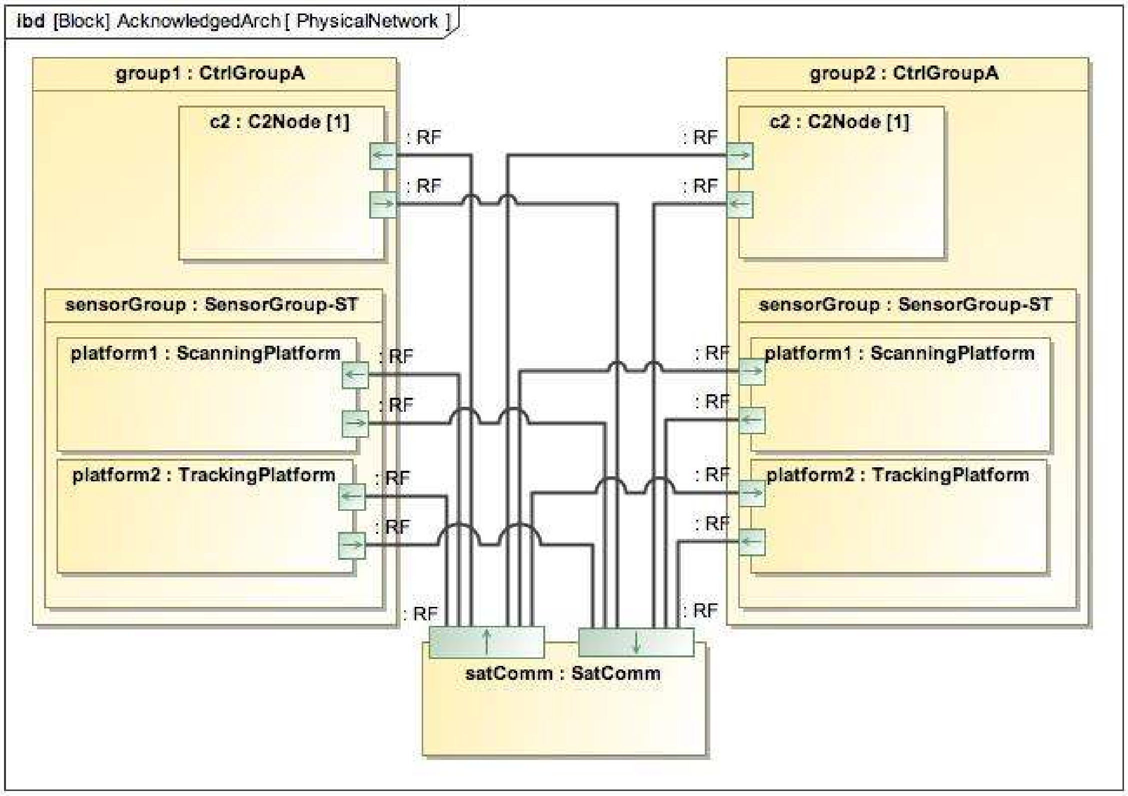

ibd [Block] AcknowledgedArch [PhysicalNetwork]
group1 : CtrlGroupA
c2 : C2Node [1]
sensorGroup : SensorGroup-ST
platform1 : ScanningPlatform
platform2 : TrackingPlatform
group2 : CtrlGroupA
c2 : C2Node [1]
sensorGroup : SensorGroup-ST
platform1 : ScanningPlatform
platform2 : TrackingPlatform
: RF
satComm : SatComm

Figure 4.2. Physical network diagram for the acknowledged SoS architecture

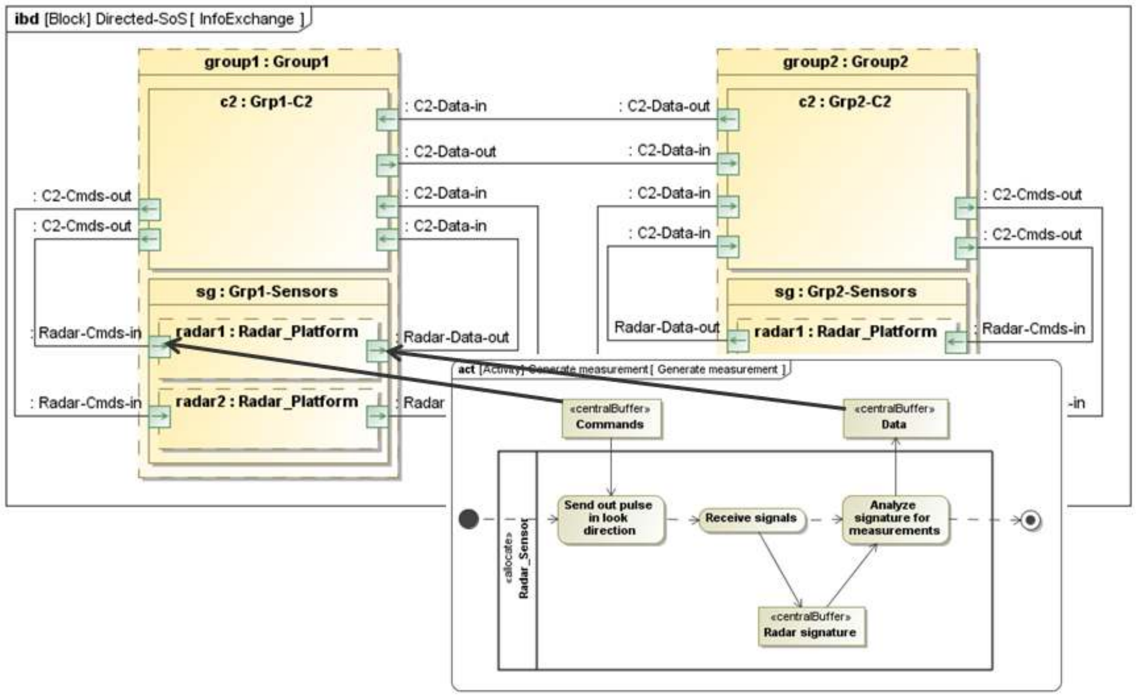


Figure 4.3. Relation between activity diagrams and logical ports

By connecting the elements from activity diagrams developed in the last chapter to the ports of systems, as shown in Figure 4.3, a relation is established between the operations a system is performing and how the data from that operation is used (i.e., where that information flows). This relation between elements from different viewpoints provides more information about the system(s) and creates a traceable path in the model that can be queried by scripts for simulation and analysis.

4.2 Port Definition

In establishing the network architecture in the previous section, a number of ports are created. Each connection between blocks in both logical and physical diagrams utilizes a SysML element known as a flowport. Flowports allow types to be set to them, letting a SoS engineer denote what logically flows over the port or what type of physical connections it supports. Port typing is used to provide further context to the port or to show commonality among several similarly-typed ports, yet this research emphasizes another purpose for them. By typing all flowports with flow specification blocks, value properties may be applied. These value properties allow for parameterized specification and analysis the same way they do for system blocks.

For the purposes of this research, ports are divided into two categories: physical ports and logical ports, to be used for their respectively cyberphysical views. Physical ports inherit value properties relating to the physical limitations of that port, such what types of data can flow in/out, data rates, etc. Logical ports then express the nature of the system and its capabilities (further shown by their connection to activities), inheriting values of how much data should be produced and at what rate. A system or its allocated function may denote a certain data production rate, but that rate will only be analyzed if it is allocated over a connected logical port and its associated physical port.

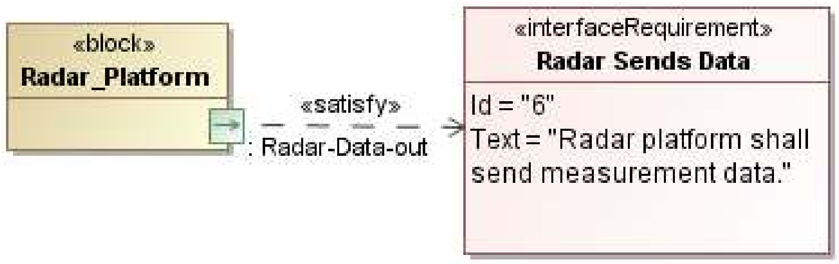


Figure 4.4. Requirement specifying the output of measurement data by a radar platform

4.3 Requirements on Ports

As mentioned briefly in Chapter 3, generating requirements or importing previously created requirements (from past SE efforts) allows for validation and standardization of system components. Using requirements for system properties was discussed as a SE task, but including requirements on ports broadens the requirement engineering scope to how systems interact. Interface requirements specify what information a system is intended to communicate, where the information is sent, and how much information should flow. To meet these needs, requirements should be connected through a *satisfies* relationship to port properties as well as system properties.

As was established in the last section, logical ports designate what type of information should be sent from a system. Connecting these ports to other system ports designates where that information will flow. Thus, a requirement on a logical flowport should designate what type of information is sent from that port. Such a requirement may be written as, “Radar X shall send measurement data” and is attached to the “Radar-data-out” port. Any connections from that port will satisfy where the data is intended to go. In the case of the Baseline architecture, this requirement is written as “Radar X provides measurement data to C2 Node Y”. Furthermore, “C2 Node Y” should have a requirement on its port denoting that it receives measurement data. Ensuring that requirements are established on both ends of a connection is referred to as requirement gap analysis and is discussed more in Chapter 5.

For physical ports, requirements specify what necessary physical connections a system needs to support. Requirements on a radar may say that it is capable of sending data over a RF connection at a speed of 200 kb/s (this is used as a default value). If the language of a requirement is formalized well enough, a user script can ensure that the model elements are matching the requirements attached to them and raise a flag any time a requirement is validated. However, this is not an inherent feature in SysML due to the fact that requirements may be written differently for separate projects. A validation script exists for the models shown in this thesis to automatically synchronize numerical information in requirements with the value properties of their connected elements (ports and systems properties).

4.4 Integration of System Models

As the goal of MBSE is to provide a single source of truth for information on a system, the scope of a SoS model focuses on the high-level properties and interactions of systems; lower levels are already established through standard SysML SE practices. A benefit of modeling is its capability to examine various levels of abstraction. Thus, it is a useful capability for SoS modeling to integrate a more detailed system model into a SoS model. A radar system model was created separate from any of the SoS models and then integrated into the Acknowledged model in order to test the ability for the SoS model to utilize the system model. The model is not meant to be a complete description of the radar system, but instead just to capture some of the key facets that may be modeled in order to best simulate the importing of such a model into a higher-level SoS model. Though factors of the integration of a system model into a SoS model is tool-based, this thesis will focus on the abilities of SysML as a language to integrate the two.

The independent radar model defines the components of the radar system in depth by including the communication antenna, sensor array, CPU, and power supply. A structural decomposition of these elements is shown in Figure 4.5 on a bdd. The

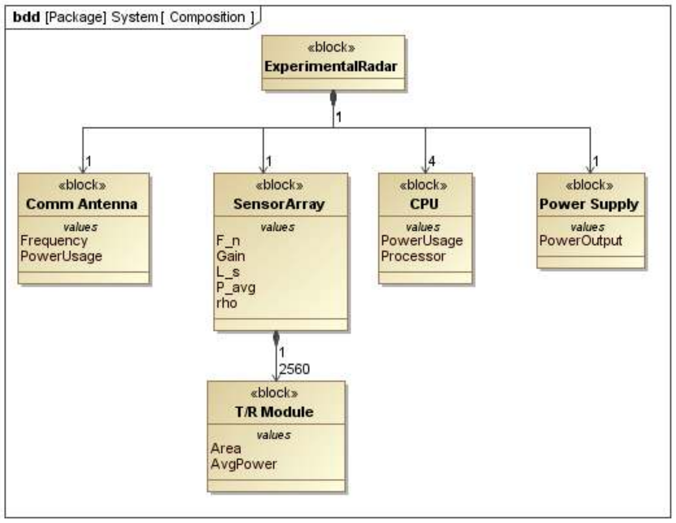


Figure 4.5. Structural decomposition of the independent radar model.

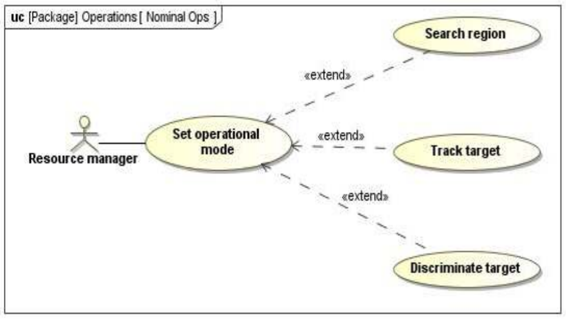


Figure 4.6. Use case diagram showing three key use cases: search, track, and discriminate.

sensor array is then decomposed into the individual transmit/receive (T/R) modules. The multiplicity on this relationship specifies how many T/R modules there are – with 2,560 being the estimated number of modules used in the UEWR radar system [24] Each component has value properties to further define it. Common radar sensor properties are found on the T/R module block – area and average power – or the sensor array block, which contains aggregate properties for the entire array. Other basic properties of components are shown on the blocks to simulate what a system model may contain (e.g., component power usage and supply output may be used in system power management activities).

The model also contains a use case diagram to capture activity sets used by the radar. This is shown in Figure 4.6, with the radar being capable of searching for, tracking, and discriminating targets. Each of these use cases maps to a related activity, shown in Figure 4.7. These activities are similar to the ones created for the SoS models but are purposefully defined differently. Without significant oversight or pattern definition from the integrating entity (i.e., the SoS modeling group), it is common that an independent system model will define things differently; thus, it is important to test what occurs when such differences are merged into the SoS model.

Once the system model is fully defined, it can be merged into the SoS model. There are various ways of moving the data from one model to another (project merging, importing the data as a module, copy/pasting, etc.) that accomplish the exact same

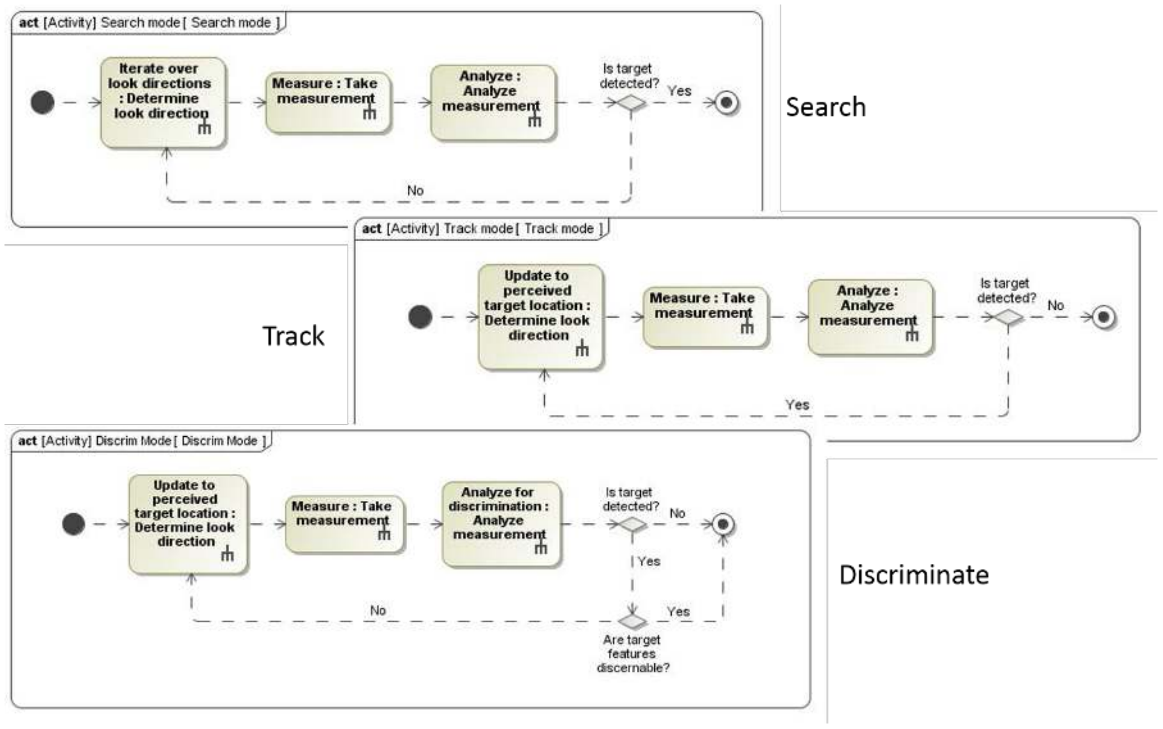

act [Activity] Search mode [Search mode]
Iterate over look directions : Determine look direction
Measure : Take measurement
Analyze : Analyze measurement
Is target detected?
Yes
No
Search
act [Activity] Track mode [Track mode]
Track
Update to perceived target location : Determine look direction
Measure : Take measurement
Analyze : Analyze measurement
Is target detected?
No
Yes
act [Activity] Discrim Mode [Discrim Mode]
Update to perceived target location : Determine look direction
Measure : Take measurement
Analyze for discrimination : Analyze measurement
Is target detected?
No
Yes
No
Yes
Are target features discernable?
Discriminate

Figure 4.7. Activity flows for key operational modes: search, track, and discriminate

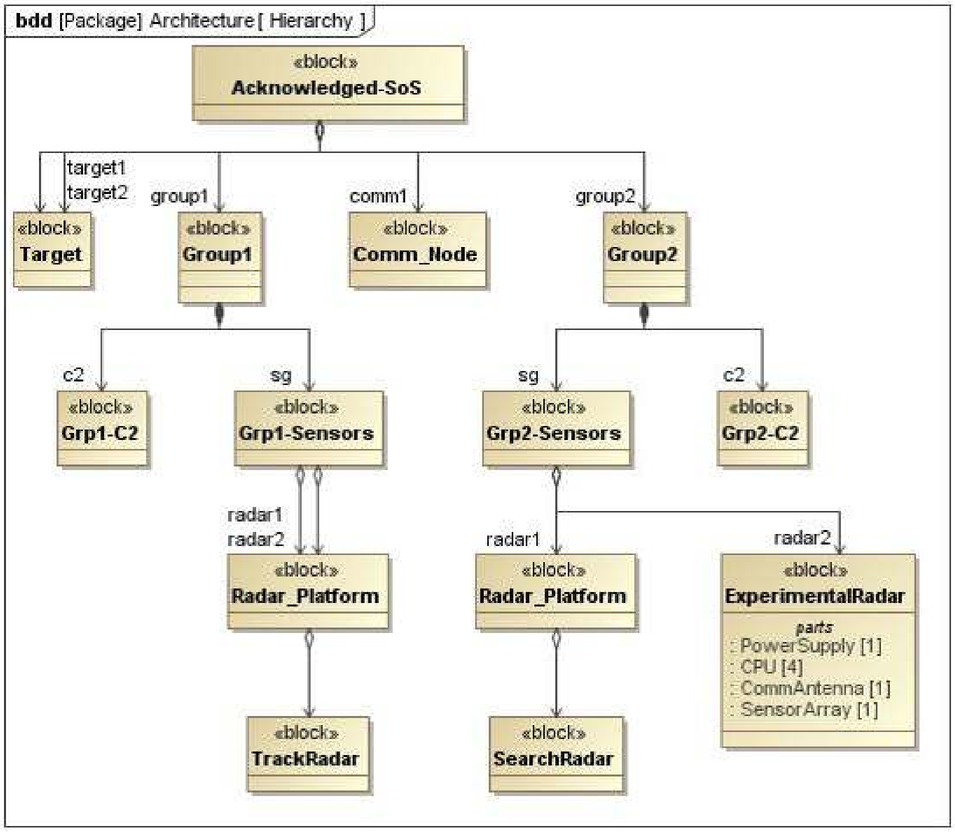


Figure 4.8. New radar model (ExperimentalRadar) integrated into the Acknowledged SoS.

task. Though these may be tool-dependent in their execution, the key element is the ability to transport the data from the system model into the same environment as the SoS model (this is an assumed tool capability). For the purposed of this test, a copy and paste of all model elements from the radar model to the acknowledged SoS model suffices.

At this point, the system model data is included in the model and remains independent of all other model objects. The process of incorporating it is similar to the process used to create the SoS model. First, the independent radar model must be added into the structural hierarchy of the SoS by removing and replacing (or refactoring if the tool allows) a previously existing radar platform block. The resulting

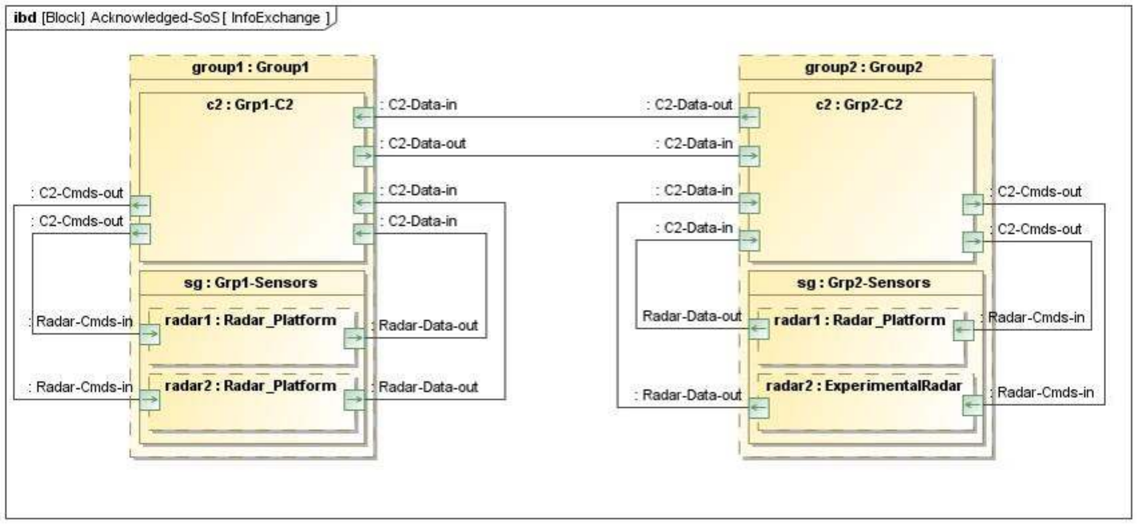


Figure 4.9. Acknowledged SoS logical network with a radar platform replaced with an experimental radar system.

hierarchy is shown in Figure 4.8. This is only done for a single radar platform for simplicity and could be done for any number of radar platforms by repeating the process. However, more steps are necessary to meet the definition of the baseline architecture.

Earlier, the generalization relationship was used to create properties on several blocks without having to define them separately. This same process must be used with the imported radar system and can also be used to ensure that the new block has important properties used in other parts of the model. It is important to note that inheritance of attributes within SysML creates new properties/ports for the block and does not overwrite any existing properties/ports, even those with the same names (part properties do not create global naming conflicts). For this example, the new ExperimentalRadar block inherits from the RadarPlatform block in order to automatically create properties associated with radars and the ports needed to network with other systems. Furthermore, the SensorArray component inherits sensor properties from the RadarSensor block.

The new radar must then be incorporated into the logical and physical networks (refactoring in some tools may keep these relations intact). For this example, it will be connected in the same manner as the radar platform that it replaced, though a new system may require/provide different interactions. Since the new radar did not specify any ports, new ports must be defined and typed or inherited for the system. The radar is then included in the network in the same way as the previous radar platforms. For reference, the resulting logical network is shown in Figure 4.9.

Since the system block uses its own set of activities, these can remain intact. The activities may be added to the activity hierarchy if desired, but it is not entirely necessary to define the operations. Relations between the activities and the newly defined ports should be created. Any requirements that exist on the radar system may be integrated into requirement lists for the entire system or kept separate depending on the needs of the SoS engineer and the organizational structure of the model. Changes to analyses and related diagrams are discussed in Chapter 5.

4.5 Networking with Different SoS Types

4.5.1 Directed SoS

The Directed SoS is logically networked so that all sensors pass data to the C2 node in their control group. The C2 node responds by sending commands to these sensors. Coordination between control groups is managed by the high-level command system, which receives data from the C2 nodes of each control group and sends commands to these C2 nodes. This network topology befits a directed SoS due to the fact that each control group remains independent in their operations and management; yet, there is still the possibility of a central management structure directly influencing the operations of either control group. These features can be seen in Figure 4.10.

The physical network for the Directed SoS, shown in Figure 4.11, routes all communications between a communication node. This is not intended to be an optimal

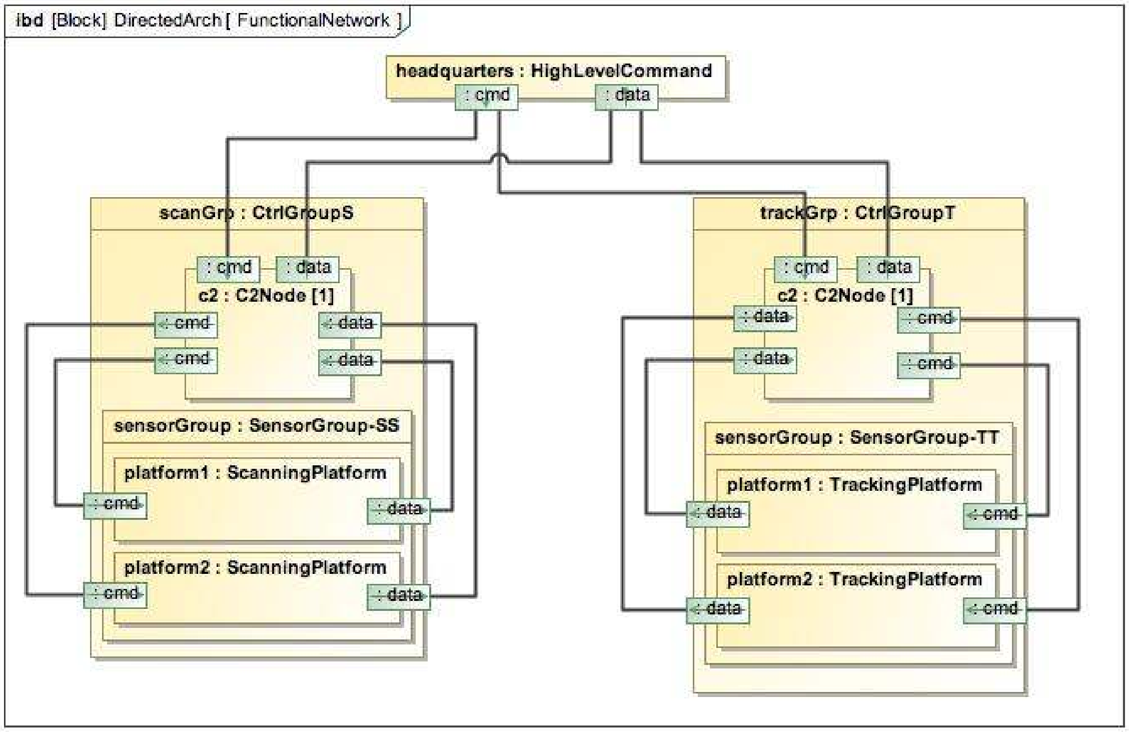

ibd [Block] DirectedArch [FunctionalNetwork]
headquarters : HighLevelCommand
: cmd
: data
scanGrp : CtrlGroupS
: cmd
: data
c2 : C2Node [1]
: cmd
: cmd
: data
: data
sensorGroup : SensorGroup-SS
platform1 : ScanningPlatform
: cmd
: data
platform2 : ScanningPlatform
: cmd
: data
trackGrp : CtrlGroupT
: cmd
: data
c2 : C2Node [1]
: data
: data
: cmd
: cmd
sensorGroup : SensorGroup-TT
platform1 : TrackingPlatform
: data
: cmd
platform2 : TrackingPlatform
: data
: cmd

Figure 4.10. Directed SoS logical network

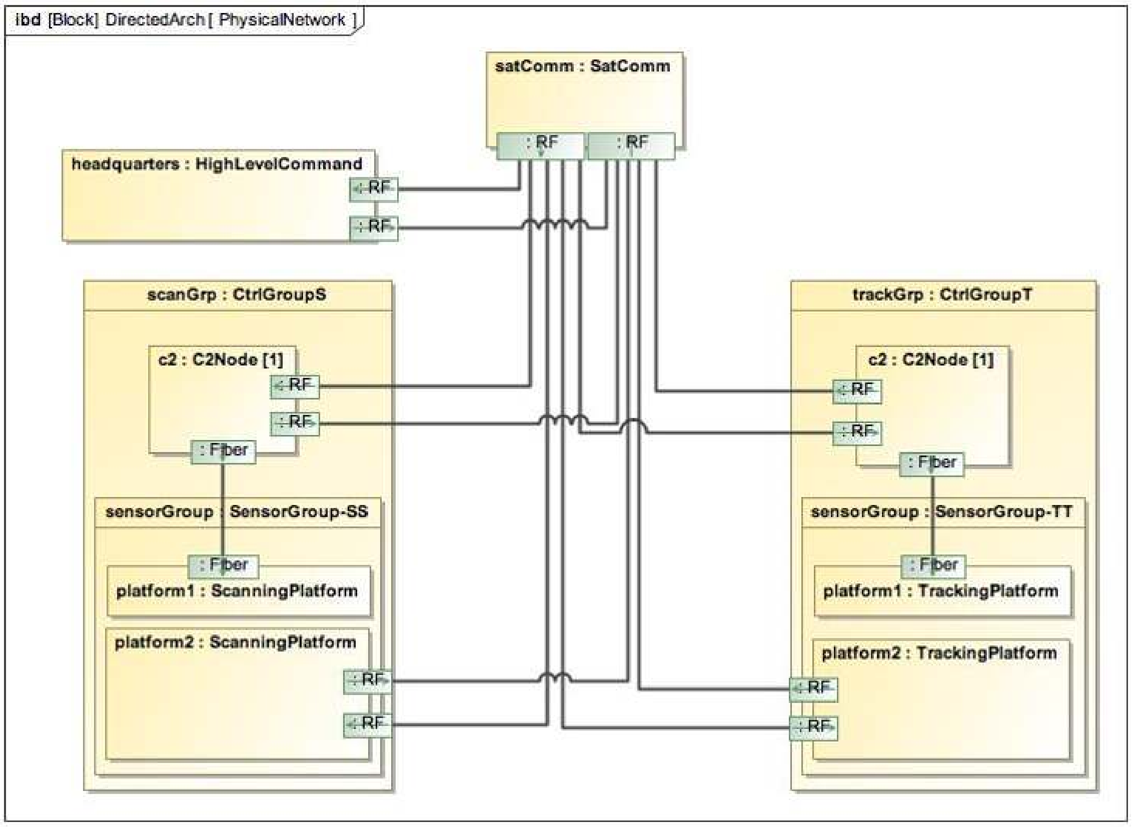

ibd [Block] DirectedArch [PhysicalNetwork]
satComm : SatComm
: RF
: RF
headquarters : HighLevelCommand
: RF
: RF
scanGrp : CtrlGroupS
c2 : C2Node [1]
: RF
: RF
: Fiber
sensorGroup : SensorGroup-SS
: Fiber
platform1 : ScanningPlatform
platform2 : ScanningPlatform
: RF
: RF
trackGrp : CtrlGroupT
c2 : C2Node [1]
: RF
: RF
: Fiber
sensorGroup : SensorGroup-TT
: Fiber
platform1 : TrackingPlatform
platform2 : TrackingPlatform
: RF
: RF

Figure 4.11. Directed SoS physical network

configuration of the network, but just an arbitrary configuration to show the networking capabilities within SysML.

4.5.2 Acknowledged SoS

The Acknowledged SoS networks are presented in this chapter in Figures 4.1 and 4.2. The only difference between these networks from the Directed SoS networks is the absence of the high-level command block. This also means that C2 nodes do not receive commands from any other systems, the coordination between control groups is now in the form of data shared between the groups. The implications of this simple change are vast; C2 nodes now are responsible for interpreting an entirely separate data set, the incoming stream of information from another control group. In a real-world scenario, this information may have differing levels of certainty on its accuracy based on stochastic variations on measurements, cyber-security concerns, or network delays. The intricacies of these changes are not the aim of this thesis, only the representation of such changes within SysML.

4.5.3 Collaborative SoS

The Collaborative SoS continues many of the network features from the other SoS types. However, the structural change of having four control groups, each with a single radar and C2 node, results in some aesthetic changes to the network. The interactions between each radar platform and its C2 node remains the same: the radar sends data to and receives commands from the C2 node. The logical interactions between C2 nodes remains the same as it was in the Acknowledged SoS; each C2 node sends data to the other C2 nodes. Figure 4.12 presents the logical network for this SoS type. The primary changes occur in what restrictions are placed on the physical network in order to create a competitive environment.

The physical network for the Collaborative network features two departures from the other physical networks. First, to simulate higher complexity in the radar sys-

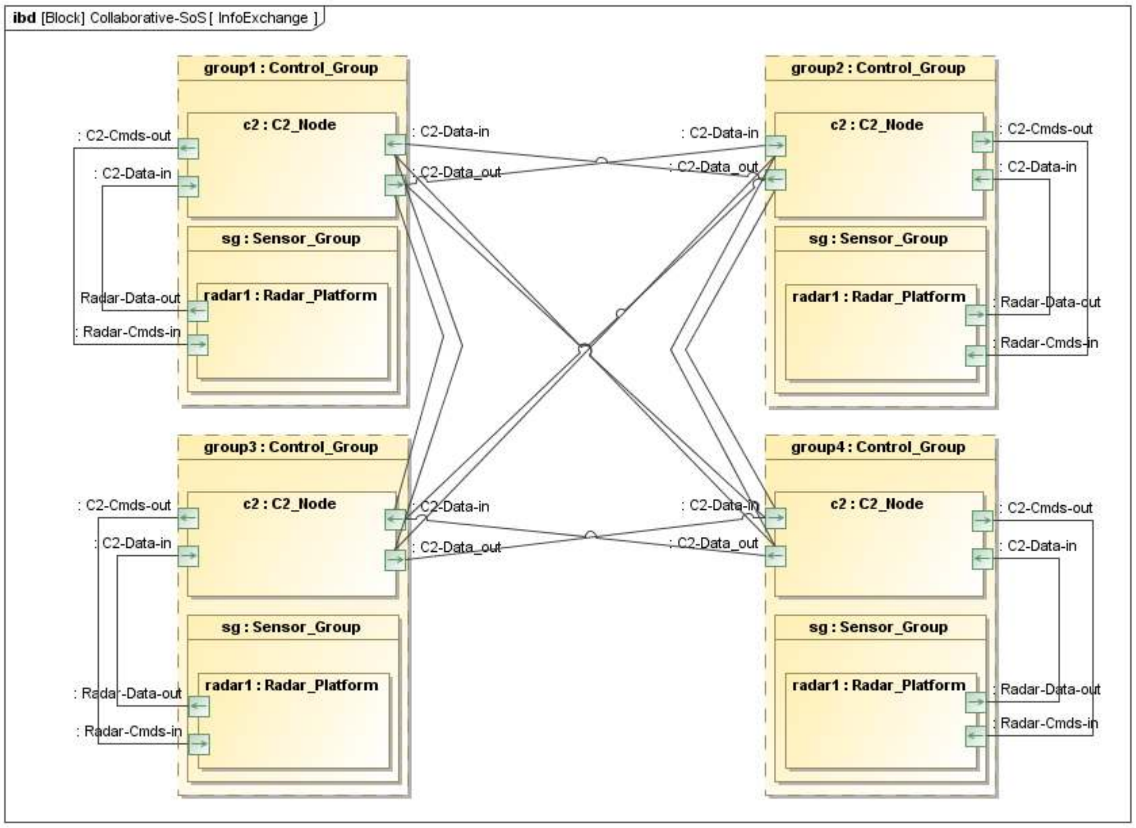


Figure 4.12. Collaborative SoS logical network

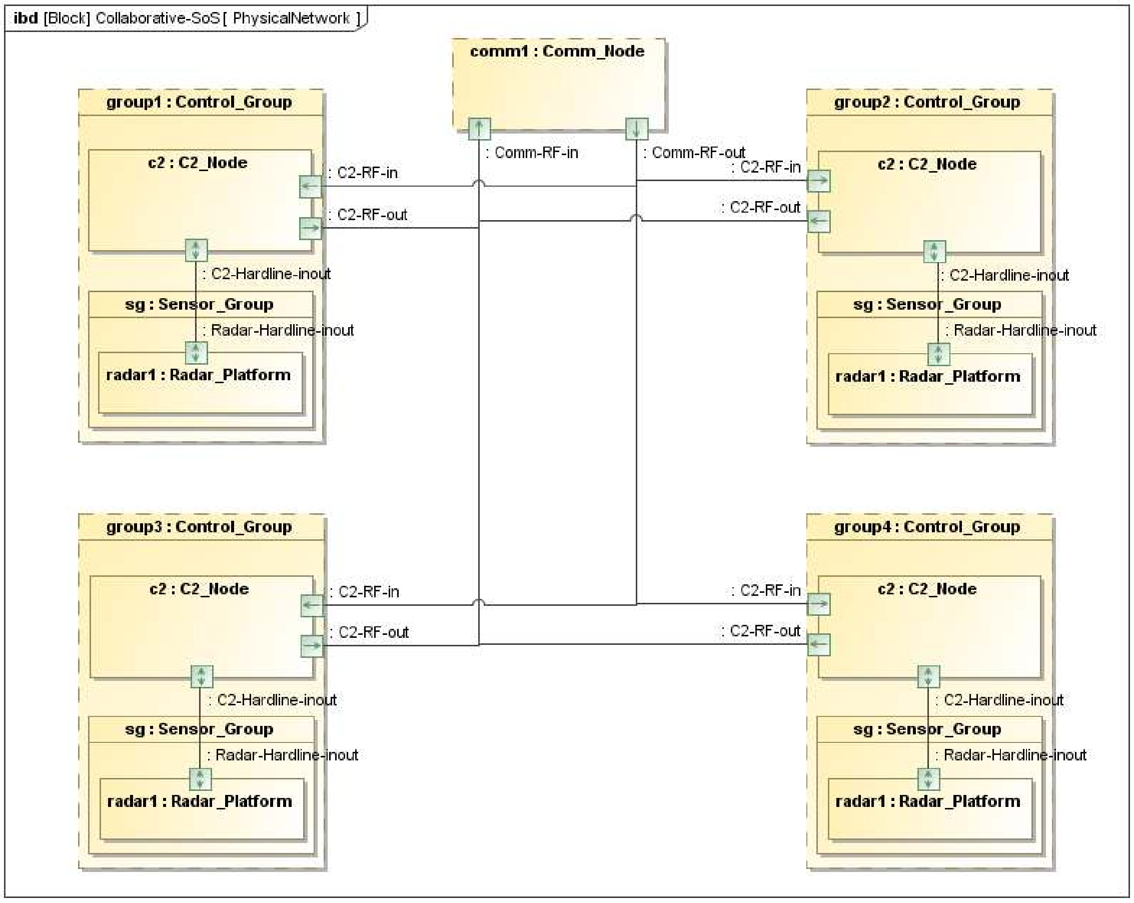


Figure 4.13. Collaborative SoS physical network

tems in this SoS, the C2 nodes are collocated with and hardwired to a single radar system. This is one method of showing the control functionality existing on the same system. Another method of showing this would be to remove the C2 node entirely and allocate its activities to the radar platform. This option is not done because it implies large changes to the underlying structure of the base radar system, namely in the computational strength of the system. Such a change to the system would require a different set of properties, a redefined set of subsystems (e.g. CPU, avionics), and possibly different port properties to convey increased data rates.

The other primary difference in this physical network are the requirements levied on the ports of the comm. node. In order to best capture the features of a collaborative SoS, focus is placed on the interface rules between the systems. This is due to the fact that directly influencing the systems in this type of SoS may not be possible. Furthermore, to simulate the voluntary inclusion of systems in the SoS, the rules governing the flowports, featured as requirements in the model, specify what a system (in this situation a single control group) must do to share data with other control groups. One requirement is placed on the RF-in port that specifies a lower limit data rate of transfer that must exist for a system to be "included" in the SoS – to be on the list of systems that receive data. Another requirement is placed on the RF-out port that states that a system must be "included" in order to be sent data. These two requirements combine to turn the comm. node into a filter of sorts, only allowing data to flow between systems that contribute data. This scenario enforces both the voluntary inclusion and interface control aspects that distinguish a collaborative SoS. All of these requirements are incorporated into simulations and analyses run on the model.

4.6 Conclusions

4.6.1 How are alternative configurations, system sets, networks evaluated in SysML within the SoS scope?

SysML can evaluate alternative configurations easily because of its extensible nature (i.e., less and more stringent interactions between elements can be defined by the user within the language). However, when dealing with SoS's, specific precautions must be considered to create a model that can be adapted, rather than recreated, every time something changes. As discussed previously, there are precautions to be taken when allocating functions to system blocks.

There must also be caution when defining the interactions between systems, which are prone to changing as the SoS evolves: new systems come on/off-line, systems change their operations, different data is required in different places. To allow for this adaptability within the model, two major steps are taken. Physical and logical views are kept independent of each other to allow for the types of interactions between systems and within what medium those interactions occur to change without disrupting each the other. Secondly, flowports are established with flow specification types to allow for both categorization and specification.

A key to allowing for the adaptability to the model to changes in architecture defining is to develop internal methods for recognizing and accounting for these changes. Here the use of specific modeling patterns comes into play. Though a pattern has been used for this thesis to broadly capture the ability to model a SoS, SysML allows for the definition of unique relations, allowing an infinitude of patterns to be defined. Some real-world applications may use a more stringent pattern to meet their specific needs, and for such a case, a process for evolving such patterns is a necessity.

4.6.2 How does SysML allow for SoS evolution?

To examine evolution of a SoS in SysML, each intermediate state of the SoS must be examined separately. To do this, the model is set up in a way that allows for different configurations of systems to be built easily. Any evolution internal to an α-level system is executed by a change to its value properties – the parameterized values that define the systems quantifiable properties – or part properties if the system is defined further within the model via internal elements. Higher-level systems are evolved by evolving their constituent systems and those system interactions. In order to change system interactions, two groups of model elements may be adjusted without altering the system: the system ports that define their interaction capabilities and the connections between said ports. Finally, functional changes between SoS states are executed by rewriting activities and their flows or adjusting how those functions are allocated. These provide a description of how a SoS may evolve while maintaining the same composition of systems, but this does not cover the addition or removal of systems.

New systems that are already modeled can be included by adding another composition relationship to the top-level domain block. For example, the changes from the Acknowledged to the Collaborative architecture sees the addition of two more control groups, each with a C2 node. This addition is done easily due to how the system blocks are established. The only reworking of the model required is to include the systems within both the physical and logical networks. To remove systems, only the composition relationship must be removed. Addition or removal of new types of systems require similar changes to be reflected in the base SoS definition, while any other changes should maintain the same redefinition pattern of the base SoS elements.

Of the different ways to evolve a SoS – changing system properties, system interactions and/or functions, and adding/removing systems – some are more easily applied in SysML than others. As stated before, changes to system properties can be evaluated quickly and easily by using generalizations and altering value proper-

Table 4.1 Overview of SoS Architecture Evolution tasks in SysML

SoSE Task	SysML Elements or Diagrams	Directed SoS	Acknowledged SoS	Collaborative SoS
System Interactions & Network Definition	Ports on blocks with connectors shown on ibd	Separate definition for physical and logical networks	Separate definition for physical and logical networks	Specifying the requirements on ports is key to defining system interactions
Integration of System Models	All elements/diagrams used in describing a system model may be imported	Imported systems are attached similar to creating a new system	Imported systems are attached similar to creating a new system	Imported systems are attached similar to creating a new system
SoS Evolution	All elements and diagrams, with emphasis on networks (ibd)	Evolution through direct changes to the model or by examining stable state(s) under a separate domain block	Evolution through direct changes to the model or by examining stable state(s) under a separate domain block	Evolution mostly through changes to the interaction rules of the SoS (requirements on the network)

ties. Changes to the system interactions (i.e., re-networking) require some work in restructuring the connections between systems. This is either done as a change under an existing domain block (permanently changing the model) or under a new domain block for side-by-side comparison, which can be easily copied (in most, if not all, tools). Similar changes are required to add or remove systems. The only evolutionary behavior that requires a permanent change to the model are changes to the allocation of functions. Any changes system behavior or new systems must reallocate these functions with the same method as they were initially allocated.

5. SOS PLANNING FOR UPDATES & IMPLEMENTATION

The goal of this chapter is to plan for SoS updates and implement those plans. In order to create a plan for updates, a thorough review of the current and possible future architectures must be conducted. This is where the latter half of the SoS analysis step in the Wave model occurs. Much of the focus of this chapter will be on presenting the analysis methods that are available within the SoS model and how those may be used to plan and implement updates to the SoS. Emphasis will be placed on the analysis of systems, interfaces, and the SoS as a whole. A discussion of how well SysML addresses the analysis needs of the SoS follows.

5.1 System Analysis

System analysis can be performed through the use of parametric diagrams. As indicated by the name, parametric diagrams use the parameters of constituent systems as inputs to a set of equations, defined by the model. For the distributed sensor model used throughout this research, determining the signal-to-noise ratio that a radar sensor experiences during a measurement on an target is important. Signal-to-noise ratio is a basic measure of how clearly a radar perceives its target. The signal-to-noise ratio (SNR) between a sensor and a target can be found by solving the radar range equation:

$$SNR = \frac{P_t * n * \tau_p * G_T^2 * \lambda^2 * \sigma}{(4\pi)^3 * k * T_o * F_n * L_s * R^4} \tag{5.1}$$

Equation 5.1 utilizes several properties of the radar sensor, including average power transferred P_T, number of pulses integrated n, pulse width τ_p, transmitted gain G_T, wavelength λ, receiver noise F_n, system loss L_s, and operating temperature

T_o. k is Boltzmann's constant [25]. The target is represented by its cross sectional area perpendicular to the sensor σ and its range from the sensor R. SNR allows for the derivation of variances on the estimated position of the target and the probability that the target is detected by the radar given a set probability of a false positive. These performance values are determined by the following equations:

$$\sigma_\theta = \frac{\sqrt{\frac{4}{\pi G}}}{1.4\sqrt{2n * SNR}} \tag{5.2}$$

$$\sigma_R = \frac{\frac{c * \tau_p}{2}}{1.81\sqrt{2n * SNR}} \tag{5.3}$$

Additional variables used in Equations 5.2 – 5.3 are angular and range standard deviations, σ_θ and σ_R, and the speed of light, c [25].

To perform these calculations within the model, each of the sensor and target properties must be established as value properties on their system blocks. Once this is done, a parametric diagram can be created to carry out the calculation. The ParaMagic plugin for MagicDraw includes a solver for any parametric diagrams. However, performing large calculations within parametric diagrams can become tedious. Fortunately, MagicDraw and ParaMagic (as well as other SysML tools) have the ability to feed model values into MATLAB and other scripting languages to be used as an external evaluation device. Parametric diagrams may be used to pass values into these scripts or the scripts themselves can extract values from the model depending on the tool (both are available via MagicDraw as it defines an API for writing scripts in multiple languages).

Because up to two sensors are used in a single sensor group, the parametric diagram seen in Figure 5.1 calculates both SNR values for a single target at the same time. Once this diagram is established it can be applied to any sensor group. For simplicity, the plots generated for the entire trajectory of the target are created in the same MATLAB code that calculates SNR. The model feeds these values in, so that if any values are changed within the model, the changes will be reflected in the MAT-

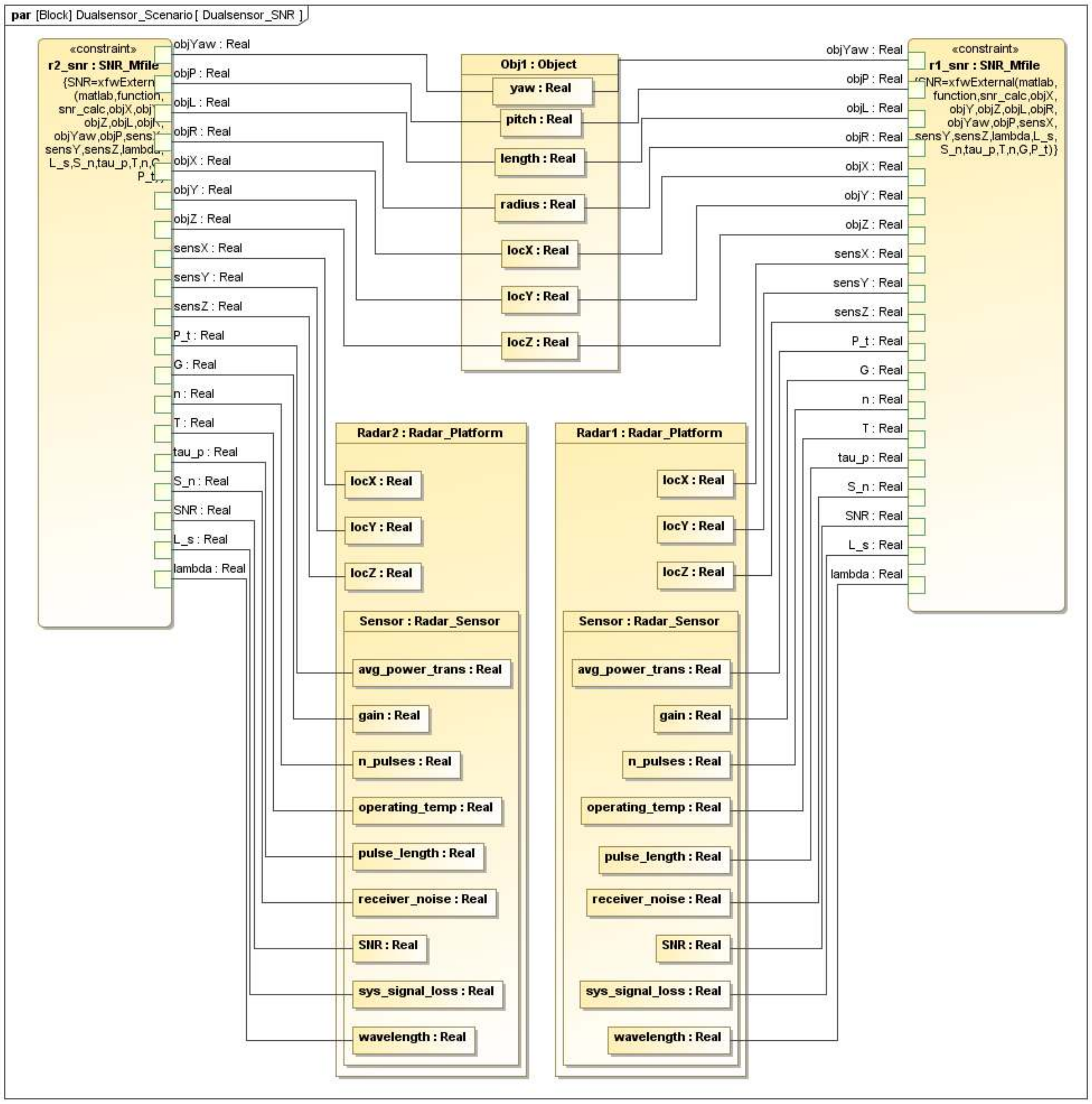


Figure 5.1. Parametric diagram showing the calculation of SNR for two sensors on a single target

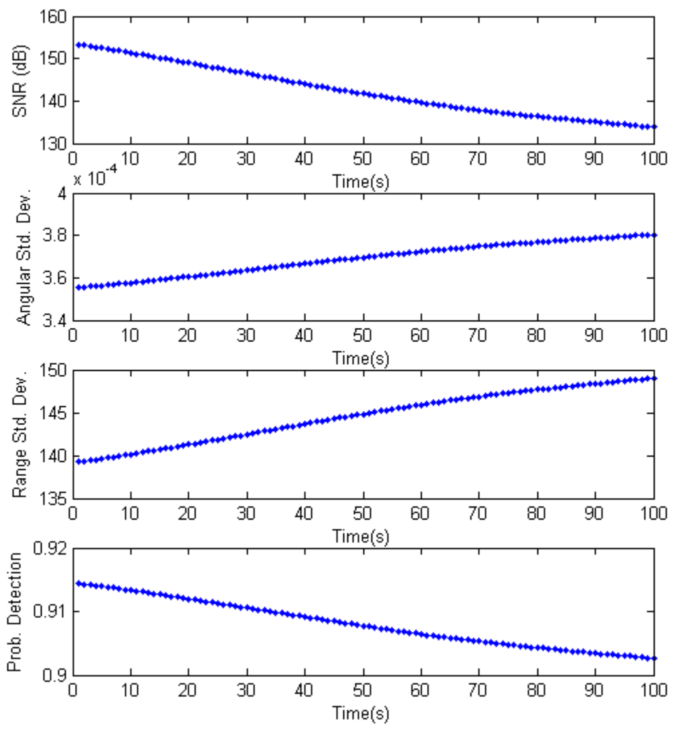


Figure 5.2. Signal-to-noise ratio and other derived performance values for a single radar and a single target

LAB outputs. Because there are two targets being simultaneously investigated for SNR values, two separate SNR value properties must be created, each set to display information about one target.

To test the performance analysis capabilities of the MATLAB script and its integration into SysML, an target is placed 100 km north of a radar sensor. The sensor is set with parameters similar to that of an Upgraded Early Warning Radar (UEWR) [24]:

Power transmitted	67 dB
# pulses	3
Pulse length	5.1E-6 s
Gain	48 dB
Wavelength	0.091 m
Operating temp.	290 K
Receiver noise	4.25 dB
Signal loss	9 dB

The cylindrical target is set with a randomized ballistic trajectory away from the sensor and a starting velocity of 1000 m/s. Drag is not accounted for, as the purpose of this research is not to develop a physically accurate trajectory simulation. The results of simulating this for 100 seconds is shown in Figure 5.2. As is expected, the SNR drops as the target is launched away from the sensor. This in turn causes the standard deviations in both angular and range dimensions to increase. Finally, the probability of detection drops slightly throughout the simulation.

This same simulation is performed for each sensor (of 6) and each target (of 2). The SNR values for the entire simulation are averaged and reported back into SysML to be used in aggregate performance analysis. Furthermore, by passing these values back into the model, they can be used in comparisons against other model architectures at a later point while maintaining the single source of truth that is desired throughout this process. Though it is possible to manipulate arrays within ParaMagic, it remains much easier to do those calculations within MATLAB.

5.2 Interface Analysis

Having demonstrated the ability to analyze the systems themselves, the next step in analyzing the SoS is to examine the interactions between systems. As mentioned in Chapter 4, the ports used in this research are specifically typed to allow for properties to be assigned to them. Physical ports are provided with maximum allowable bandwidth and excess bandwidth value properties, and outgoing logical ports have a data rate value property. Using parametric diagrams, the amount of data flowing through each commline is calculated and each utilized physical port can solve for its excess bandwidth.

Bandwidth calculations are created entirely in SysML without the utilization of external software (e.g., MATLAB), as inherent solving tools in MagicDraw allow for acausal solutions to problems. Therefore, it does not matter which variables within the calculations for bandwidth are found, as long as a solution can be determined (i.e., the resultant equations are solvable). Despite the simple nature of the calculations, it is important to take note of the methodology behind creating these parametrics. This same approach can be used to solve for the summation of data transfers in a system or to determine the maximum available data transfer over a line if others are already determined.

All information required for bandwidth calculations is gathered from the flowports of the utilized blocks. Each port that utilizes a physical connection is set to inherit properties from a single interface block. This allows for any value properties of that interface block to be applied to all inheriting blocks (via a generalization relationship). Such properties may be the bandwidth of the connection, the maximum number of connections allowable through the port, or any matter of quantifiable property. For the calculations used in this research, only the bandwidth of the port is required.

Similarly, flowports that are used for logical connections inherit values pertaining to the information that flows through that port, which may be specified by the flow specification or the activities allocated to the port. These values may describe the maximum amount of data sent over a port, the rate of data that flows over that port, or other quantifiable properties pertaining to data flows. For the sake of simplicity, this research focuses solely on the data rate of all logical ports.

Having all ports established with properties, the bandwidth calculations are set to combine all data going over a single port and analyze how much excess bandwidth that port still holds. Though these are moderately simple calculations and could be accomplished with an adjacency matrix, SysML provides the ability to pull information from various places in order to perform these calculations and also allows for acausal solving. If a low-level system or component has a design change that sees its data production increase or its bandwidth limitations decrease, these changes will instantly propagate throughout all other calculations in the model. This single point of truth capability can be crucial in avoiding errors in system design as it immediately shows the effect of the design on the entire SoS.

Via scripts it is capable to take the result of this analysis and apply visual changes to the existing diagrams. For example, a script is written to color code the connections between various blocks based on their bandwidth capability and usage This script examines the excess values from each port and colors a connection green if it has excess bandwidth, yellow if it is perfectly saturated, or red if the desired data rate over the line is greater than the allowable bandwidth. This coloration is also performed on ports, due to the fact that a single port (e.g., the communication node's RF out port) may service a number of different physical connections. The visual aspects of a model are easily altered by its inherent values because the two are so closely intertwined.

Figure 5.3 demonstrates this effect in process. Here, the communication node in the baseline architecture established with a high bandwidth on its RF-in port, but a low bandwidth on its RF-out port. All ports on the radar platforms and C2 nodes are producing and receiving data under their maximum bandwidth on those ports. However, the RF-out port on the comm. node is in violation – producing more data than it can send. It is colored red along with all connections to that port.

This visual representation is useful in quickly conveying the validity of a specific network design. If the design space is open to a number of ideas, and the rest of the model exists already, a SoS engineer may wire together various network options for analysis. Each can be tested quickly and glanced at to determine where the network structure is valid for the information needs of the systems. Similar trades can be examined in the logical network structure as well, to see if placing functionality at a different level may alleviate network issues.

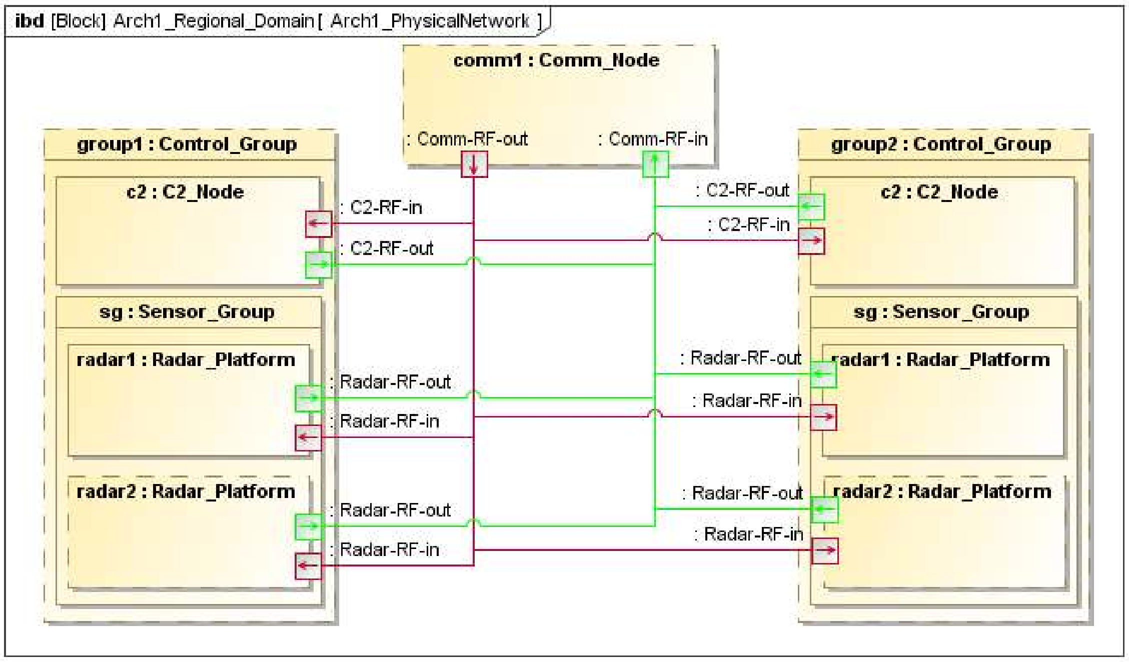


Figure 5.3. Physical network diagram displaying the state of each physical connection

5.3 Requirements Analysis

As has been previously discussed, the use of requirements provides a list of the key features that must be fulfilled when developing a system. These requirements are satisfied through elements in the model – focusing on system properties, flowports, and the connections between them. Requirements provide systems engineers with a checklist with which to ensure a system is engineered as intended. Requirements analysis in SysML functions as both a check to ensure the model is accurately portraying the system or SoS and that the requirements between systems are coordinated. This is important when dealing with a SoS, where systems are often developed under differing management structures and possibly with little coordination between interacting systems' teams.

As mentioned in Chapter 3, systems requirements are satisfied by the properties of system blocks in the model. A validation script can ensure that these stay coordinated with any changes in system updates (assuming that a system requirement may change outside of the SoSE process). Any changes in these requirements must be updated and accounted for within the model by the SoS engineer or a systems engineer responsible for modeling their system. Therefore, the key aspect of requirements engineering for SoSE lies in system interactions. Chapter 4 discussed applying requirements to ports, their properties, and the connections between them. By using this modeling infrastructure, useful analysis is generated for SE groups focusing on system interfaces.

The high-level approach of SoSE allows for a SoS engineer to analyze the requirements on multiple interacting systems in tandem. Thus, a requirement on a port specifying the generation of data must have a parallel requirement on the port that receives that data. This gap analysis is done in SysML either with a script that queries each port and its connected port and then examines both for related requirements or by generating a table that displays the connections between interface requirements. The former method may be quicker in simple cases, but when interface requirements get more complex (e.g., multiple value properties, multiple layers of information exchanged) it is best to manually examine a table of the ports and their requirements or use an automated method to examining the differences.

5.4 System Integration & Aggregate Analysis

In Chapter 4, an independent radar system model was imported to examine the capabilities of integrating existing and more detailed system models into a grander SoS model. Beyond what was discussed previously, there are also implications on the analysis aspects of the SoS that must be addressed when importing a different system model. Much of this deals with how different aspects of the model are specified.

As previously discussed, it is often useful to use generalized blocks to inherit properties to blocks that require the same properties. This can also be a useful tool for standardizing an external model to something more easily digestible in the SoS model. Though the ExperimentalRadar block, which acts as the top-level block for the radar model, already contains some value properties that are used in the SNR analysis on the existing radar blocks, some of these values are defined differently. These alternatively-named properties are equated to their identical value properties inherited from the RadarPlatform and RadarSensor blocks through a parametric relation. The new radar is now able to be used in any of the pre-established parametric diagrams. Port properties are also inherited for any ports that have been inherited from a generalized block.

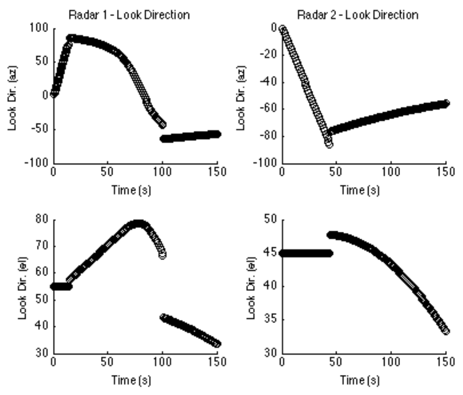
5.5 SoS Analysis

To further engage the model, querying of elements can improve the quality of simulation. In each SoS type, a simulation of sensor tracking is created. The overall goal in each SoS type is to aggregate the highest quality and highest quantity of measurements on each target. Each SoS type accomplishes this in different ways based on their set of operations and structural properties, stressing the key features of that SoS type. In every SoS type, the base level sensors are capable of searching for targets and tracking the targets that they find. Sensors are also capable of accepting commands from C2 nodes to change their behavior. The nuances of exactly how a sensor decides when to search and track or when a commands are sent from a C2 node change with each SoS type. The simulation is created by querying the SysML model for its operations procedures (i.e., at what threshold to task to a different mode) and the communication structure between targets. Calculations are performed in MATLAB.

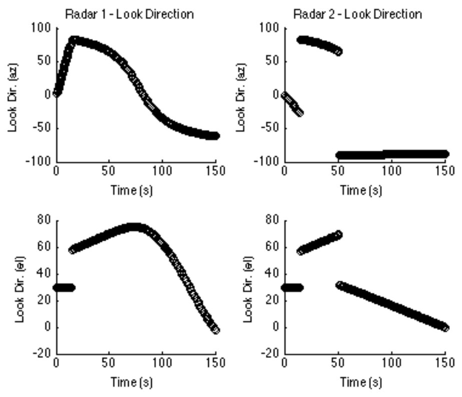
5.5.1 Directed SoS

The Directed SoS is composed of two control groups, each with two sensors and a C2 node, with a high-level command system functioning as the coordinating element between the two groups. This commander takes in the data from each control group and may provide high level commands to either. Each control group has a SNR threshold per target to determine how to task sensors. Control group 1 focuses on scanning and has a low threshold that when met, will cause the C2 node to retask the sensor into scanning again. Control group 2 focuses on tracking and has a higher threshold set for when to switch targets. Sensors in this group will also be tasked away from a target if the SNR values drop too low. The handoff between when the scanning group meets its threshold and when the tracking group picks up a target is coordinated by the high-level command.

Figure 5.4 shows sample results for the Directed SoS simulation. In this case, both sensors in control group 1 scan an area until a target is found. The first sensor in control group 1 tracks this target, while the second sensor continues to scan due to the control scheme of this group placing larger emphasis on scanning for new targets than tracking found targets. Since the tracking group does not have any targets at the start, they are tasked by the high-level command to also track the target found by the first control group, despite it being below their standard threshold for tracking. This group continues tracking the target after the scanning group returns to scanning.



(a) Control group 1 look directions



(b) Control group 2 look directions

Figure 5.4. Look directions for all sensors in the Directed SoS simulation

The scanning group then finds a second target and begins tracking it. Once this target hits the control group's threshold, they return to scanning and the high-level command tasks control group 2 with tracking the new target. Since the first target has yet to reach its upper SNR threshold, the C2 node for this group only tasks one sensor to track the second target.

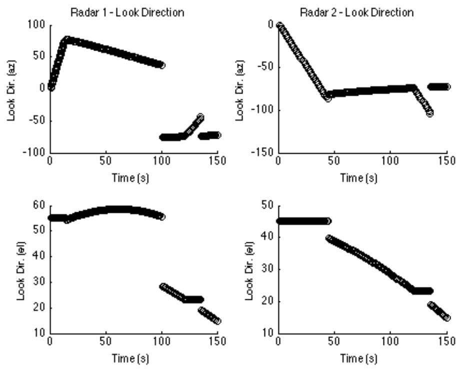
Overall, this simulation shows the capability of a high-level command system – one of the key features of a Directed SoS – to coordinate and control constituent system groups. The method of control is basic and leaves some management capability to the systems below it, as evidenced by the C2 node determining whether to task one or both sensors to track a target. Sensors in this SoS are not capable of much decision making outside of determining their own look directions and tracking targets.

5.5.2 Acknowledged SoS

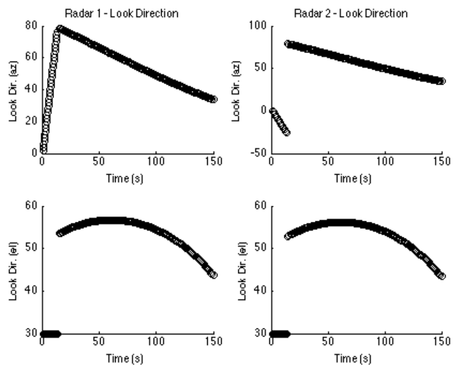
The Acknowledged SoS functions on many of the same principles as the Directed SoS, but without the use of a high-level command system. In this SoS, the two control groups coordinate directly via their C2 nodes transferring information. Neither C2 node sends commands to the other control group, only data is transferred; this is to allow each control group its own decision making. A scenario is established to stress this managerial independence by showing conflicting operational principles in a control group.

The control groups are again set so that control group 1 focuses on scanning, and control group 2 focuses on tracking. Both use the same SNR threshold principles as in the Directed SoS. However, instead of a command being issued for the hand-off between low SNR tracking and higher SNR tracking, data is continuously sent about the targets being tracked. The control groups are also aware of each others thresholds.

In the simulation described in Figure 5.5, the results of a simulation with the Acknowledged SoS are shown. The same initial conditions were used from the Directed SoS, so the simulation begins the same way. One target is detected and tracked by



(a) Control group 1 look directions



(b) Control group 2 look directions

Figure 5.5. Look directions for all sensors in the Acknowledged SoS simulation

one sensor from control group 1 while the other continues to scan. Control group 2 joins in this endeavor after receiving data about the target and while not having any known targets of its own. The target is tracked until it reaches the scanning control groups SNR threshold, at which point those two sensors begin scanning again.

After the second target is detected and tracked to the first control groups threshold, the two sensors in this group again return to scanning. In the Directed SoS, the second control group was tasked to begin tracking this second object; however, this simulation is set so that until the first target reaches its upper threshold, the sensors will not be retasked. Thus, the second target gets picked up again a few seconds later by one of the scanning sensors and, having dropped below the SNR threshold again, is tracked shortly. The second control group never retasks to track the second detected target.

The goal of this simulation is to emphasize the separation of control between control groups. Each operates on its own principles with the overall goal of increasing the cumulative SNR of each target to a high level. It is engineered not to achieve the best performance for the sample case shown, but to examine what occurs when the two groups follow their own operations procedures over what may be best for the entire SoS. The absence of a central governing body shows a lack of cohesion between the groups – this is an intended feature for this example.

5.5.3 Collaborative SoS

The Collaborative SoS operates based on the principle of voluntary inclusion in the SoS. Each radar sensor is split into its own control group with a collocated C2 node and coordinates through the comm. node with all other control groups. The goal of each control group is to detect and track as many targets as possible to a SNR threshold. They coordinate data with each other in return for the other systems' data, a mutually beneficial circumstance for all groups involved. However, protocols are put in place to remove a control group that does not contribute data (does not have any

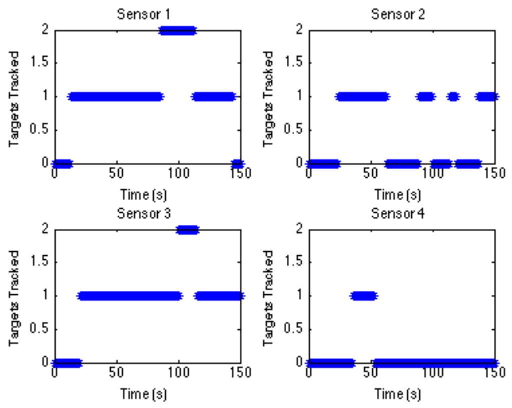


Figure 5.6. Measurement rate plots for a sample Collaborative SoS simulation

tracks to follow). This lower limit is a fixed offset from the average measurement rate of all contributing sensors – avoiding the situation where all groups are isolated because no targets are found. After a sensor is removed, it will begin receiving data again after sending enough information to surpass the cutoff limit.

Figure 5.6 shows the measurement rates of all control groups in the Collaborative SoS simulation. Limitations are placed on the field-of-regard (viewable azimuth and elevation angles) and range of all sensors involved in order to ensure that some sensor groups may not be able to see the targets at a certain time. In the simulation shown, the randomly generated targets stay in the vicinity of sensor groups 1 and 3 for much of the time. Sensor group 4 rarely sees a target and thus is cut off from receiving data. It never recovers from this due to the lack of visible targets. Sensor group 2 comes close to the lower limit measure rate, but manages not to surpass it.

The use of voluntary inclusion and limitations on contribution are both key features of a Collaborative SoS. There remains the overall goal of increasing the track quality on a set of targets, but the means by which this occurs are much different than in the other SoS types. The focus of this simulation is to show the capability of utilizing interface requirements to create rules on interactions that can then be analyzed via simulation. The contribution lower limit exists as a requirement on the data-in port for the comm. node. This captures the feature of removing a system from the group, whereas the capability for a system to be re-added exists in it overcoming the threshold.

5.6 Updates & Implementation

Having examined the systems, interfaces, and aggregate SoS, a SoS engineer can plan for updates to the SoS. Trade studies can be performed within the model through the use of either external tools (e.g., MATLAB, STK) or internal parametric analysis. Such trade studies allow for SoS engineers to examine the design space before proceeding with the implementation of changes to the SoS.

In the case of the SoS simulations described in this chapter, there are various key ways in which the information created can be used to plan for updates. With the Directed SoS, a key capability is to ensure that the high-level command is capable of communicating with the constituent systems. Thus, the network analysis discussed previously is of high importance. Since much of the control is set at this centralized point, it remains pertinent to ensure that communications to that point stay intact.

For the Acknowledged SoS, the key emphasis in planning updates to the SoS is to ensure that the behaviors of the systems are as cooperative as possible. In the example simulation, this would be done by adjusting the thresholds at which C2 nodes will change their sensor behavior. Lowering the tracking upper threshold can fix the issue of that control group not picking up a target ready for precision tracking.

The Collaborative SoS is dependent on the rules of interaction that are set in place. With a collaborative SoS, control over the evolution of the systems involved may not be possible, but the rules and protocols by which they interact are controllable. Thus, changes to the rules for inclusion, how data is shared, or when a system should or should not be included in data sharing are important factors to examine. Studying these will allow for updates to the SoS.

During the implementation step, the major role of a SoS engineer is to oversee the process and begin planning for the next analysis step. Testing, evaluation, and deployment of systems all occur during this step. Though the SoS engineer facilitates the process, there is not much to be done within SysML.

5.7 Conclusions

5.7.1 What analysis methods does SysML provide for the SoS and how well does SysML execute said analysis methods?

SysML allows for the planning of updates to a SoS by providing a number of options for evaluation. Within MagicDraw, there exists tools for doing parametric analysis and performing trade studies based on that analysis (through the ParaMagic plugin). Furthermore, SysML's definition allows for analysis to be performed at any level of abstraction within the SoS, with lower-level performance characteristics feeding upwards into higher-level analysis. This is a key capability within SysML that allows for useful analysis of a SoS; multiple levels of abstraction are captured and analyzed within the same model. By modeling each system at a low level and then developing the interactions that system has with other systems, the emergent properties of the SoS can be realized and examined.

System performance analysis

The performance analysis examined in this thesis focus on a primary performance characteristic of radar systems – the signal-to-noise ratio. To analyze this, parametric constraints are used, feeding in key operating parameters of a radar system and the target that it is measuring. To add an extra degree of detail, MATLAB is used to generate trajectories for all targets, allowing analysis over numerous data points. Analysis of parameters of systems in SysML is basic since the information is contained in a single localized repository that may be analyzed by MATLAB or other scripting tools.

However, handling arrays in SysML is not nearly as simple as pushing the same analysis into a MATLAB script. Similarly, one would not try to perform detailed calculations on the stress and strains of a system component through parametric constraints; such an analysis would best be calculated though structural analysis software. These more complex analysis types are best performed in external software, many of which have plugins that will directly connect into the software as ParaMagic does with MATLAB [10] [11] [26]. The key capability of SysML relies in capturing and relating the key elements of a system or SoS in a centralized format and providing the interfaces to perform the necessary analysis via external methods. SysML was not developed as an analysis tool but rather as a method for performing tasks to support SE. Similarly, it should retain that role within SoSE, providing the framework for developing, establishing, and interacting with external tools for the analysis of a SoS.

SoS Analysis

Complete analysis of a SoS is best done through a simulations and analyses of the network of systems. At the moment of this writing, the simulation tools within MagicDraw (the modeling tool used throughout this thesis), as it analyzes SysML, are still limited in their capability. However, by querying the model through scripting capabilities (inherent to MagicDraw and other SysML tools), important elements can be examined and interpreted in various external software. The examples shown in this research pull data from each model and run it through a MATLAB script.

Table 5.1 Overview of SoS Plan for Updates tasks in SysML

SoSE Task	SysML Elements or Diagrams	Directed SoS	Acknowledged SoS	Collaborative SoS
System Analysis	Parametric diagrams, value properties on blocks and ports	Parametrics may be built into system blocks	Parametrics may be built into system blocks	System analysis focuses on the ports and requirements on the systems
Network Analysis	Parametric diagrams, value properties on blocks and ports	Parametrics allow for analysis of network properties	Parametrics allow for analysis of network properties	Parametrics allow for analysis of network properties
SoS Analysis	Entire model, often via external tools	Focus on the control structure of the SoS, specifically functional allocation and hierarchy	Focus on the functional allocation of systems and interactions between them	Focus on the network properties and rules of interaction, described by requirements

6. CONCLUSION

6.1 Systems Engineering vs. SoS Engineering

As SysML is a language built around the aspects of Systems Engineering, the transition to utilizing it for SoS engineering exacerbates key differences in the processes. As was introduced earlier in this paper, the US DoD outlined key differences in various fields between systems engineering and acknowledged SoS engineering (see Table 1.1. These differences were considered during the research for this thesis. Table 6.1 displays information on the processes and findings on the key changes from SE to SoSE n SysML for each field.

6.2 Is SysML capable of demonstrating the 8 traits of a SoS?

6.2.1 Managerial Independence

The managerial ownership of systems is reliant on the practitioner to define in SysML. It is specifiable through various different methods as well. In this research, managerial independence is described via the structural hierarchy of the SoS. This is shown through the use of control groups in the example models. Each sensor belongs to a sensor group which is then set within a control group. The control group establishes the boundary of management, as each control group is managerially independent. It is possible to establish another level of control at the regional level (the top level in the example SoS), as is shown in the Directed SoS, but it is important to limit how much control this element has. It would then fall to the practitioner to incorporate multiple regional commands in order to reincorporate the managerial independence of the SoS.

Table 6.1 Systems Engineering vs. SoS Engineering

Aspect of Environment	System via SysML	SoS via SysML	Findings
Management and Oversight			
Stakeholder Involvement	Stakeholders defined in Use Case diagrams	Stakeholders defined in Use Case diagrams, stereotyping, or hierarchy diagrams with concern for stakeholders at varying SoS levels	The difference in defining stakeholders lies in being able to establish them at different levels in a SoS. Use case diagrams are operational focused, which can cause issues when operations are allocated at different levels in a SoS. Stereotyping allows for identifying stakeholders independent of scope, but does not have much other functionality. Identifying stakeholders through hierarchy diagrams relies on the stakeholders having a hierarchical structure similar to the one defined in the SoS
Governance	Management is structured	Managerial independence among systems necessitates the definition of management for systems	The managerial independence of a SoS is captured in SysML not only by the hierarchical structure and interactions described in the model, but also the functional allocation of activities to systems. By how these two aspects of the model are done, a SoS can have any varying degree of managerial independence.
Operational Environment			
Operational Focus	Single set of activities is created and allocated to the system and its components	Various activities are created based on a Concept of Operations; activities may be allocated to many different systems	The operational focus for a SoS is on both the set of operations for each system and the objectives of the SoS as a whole. The separation of these two establishes operational independence. Capturing this in SysML relies on properly defining the objectives of the SoS and how the objectives of the systems meet these objectives; accomplished with activity hierarchies and allocation to system blocks.
Implementation			
Acquisition	Established through a Systems Engineering Plan (SEP), which may be migrated into model or hyperlinked	New systems may be modeled within the SoS or have pre-existing models imported	The ability to incorporate external system models into a SoS framework in SysML allows for system working groups to develop models independently with a SoS working group focusing on integration and interactions. However, issues may arise with different methods of representing aspects of a system or overlap between SoS elements and system elements (e.g., doubly defined ports). A model practitioner must manually resolve these issues.
Test & Evaluation	Test plans may be created through the use of requirements and related views	Testing must be done via simulation or on a per-system basis	Testing and evaluation for systems in SysML focuses on creating a test plan to be carried out external to the model, or with input from the model. Since this is rarely possible for a SoS, simulations are performed. SysML internally contains no simulation capabilities, but allows for a framework for tools to generate simulations. Some tools extract operations directly from SysML diagrams (e.g., ParaMagic, Cameo Simulation Toolkit), turning SysML into an executable language. For this research, a custom script for extracting information from the model and running a MATLAB simulation was used. Either method requires a large amount of early-development time, but little maintenance if built correctly.
Engineering & Design Considerations			
Boundaries & Interfaces	Focus on internal components via ibd's; interfaces are only used as in's/out's for that system	Establish interactions between systems via physical and logical networks, ports on systems blocks denote in's and out's	Much of the development work with a SoS model is establishing the interfaces between systems. The interface modeling methods described earlier in this thesis effectively identifies what and how data flows between systems. Built in validation ensures that either end of a connection matches in type; this validation can be extended to requirements set on those ports. With these validation methods, interface requirements can be analyzed for gaps and inconsistencies.
Performance & Behavior	System engineered to meet performance specifications; these can be verified through parametric and requirements analysis	System requirements and SoS requirements may clash; both must be analyzed for performance. This may be verified with parametric and requirement analysis as well as simulations	System performance within a SoS may be analyzed the same way as a system model, using parametrics, requirements, external tools, etc. SoS performance may require the use of simulations to determine performance levels. Furthermore, if the model is stochastic, trade studies may need to be performed to determine performance under varying conditions. The ParaMagic tool has capabilities for constructing trade studies with the use of a spreadsheet of parameters to investigate.

Other methods exist for demonstrating ownership of systems. It may be useful to use the SysML stereotype, which allows labeling and grouping of elements without creating relations to other elements, or a stereotyped dependency relationship. A stereotype can be used as a label, specifying one systems incorporation into a group. For example, if the example sensor architecture were expanded to define a set of U.S. joint-force military sensors, there may be one stereotype for the Navy and another for the Air Force depending on the sensors locations at sea or on land [27]. Similarly, one could define stereotyped relationships to either of these forces. This avoids the trouble of hierarchical organizing elements, but is entirely free in its usage, as these are orthogonal definitions to the elements they are defining. A stricter definition of ownership was used in this thesis to best test the capabilities for SysML to capture a SoS. Furthermore, this pattern allows for systems to change ownership dependent on their configuration.

6.2.2 Operational Independence

The operational independence of constituent systems is specified by the allocation of activities to system blocks. The steps specified in the previous section for functional decomposition will establish a set of possible activities. How these activities are connected depends on their definition through both decomposition and activity and/or sequence diagrams. Thus when examining operational independence of a constituent system the key question is to examine if that system has a set of activities attributed to it that are independent of other activities.

For the examples shown in this thesis, all radar platforms are capable of performing their own scanning and tracking, while relying on a connected C2 node for higher level tasking. The radars are capable of performing these independent of other systems. C2 nodes can similarly generate commands without the input of sensors; however, the quality of such commands may be lacking without the knowledge supplied from the sensors. Operational independence depends heavily on how the activities outlined for the SoS are allocated to the constituent systems of that SoS. This directly impacts what type of SoS is modeled or if the model describes a SoS at all.

6.2.3 Evolutionary Behavior

Evolution of the SoS is captured through the modeling of separate stable intermediate forms of the SoS. To do this, a baseline SoS is redefined according to the changes between the three specific forms of the SoS. Because all of the base elements and their interactions are already established, evolving an architecture is an easier task than creating one from scratch. Any change to these elements can be enacted within these configurations without the issue of altering the original form. Any further exploratory configurations may inherit from the original baseline or any of the previously modeled configurations. For this thesis, three different SoS variations are specified from a baseline definition of elements. Furthermore, an example is shown of how to incorporate new system models into a pre-existing SoS model to simulate acquisition or evolution of new systems. In Chapter 4, methods to create a model that allows for evolution are discussed.

6.2.4 Emergent Properties

Emergent properties are modeled in SysML through system interactions. These can be in the form of new activities or sequences that are only added when specific systems interact or in the form of parametric analysis (discussed in Chapter 5). The inclusion of new activities when certain systems interact is a case of designed emergence – the modeler is conscious of the properties. If complex parametric diagrams are set up for analysis, emergent properties may arise that were not previously known (e.g., communication paths are strained with the addition of systems).

6.2.5 Geographic Distribution

Geographic distribution is established through the definition of value properties that describe the location of physical systems. In the distributed sensor model, all physical systems inherit these properties (denoting Cartesian location) through a generalization relationship to a "Physical Object" block. Though it is not a necessary condition that a SoS be geographically distributed, it is a trait that is easily addressed in SysML.

6.2.6 Heterogeneity

Heterogeneity is another trait that is easily expressed in SysML. By default, every block that is created should represent a different system or component. At the α-level of the distributed sensor SoS, three different systems are established – sensors and their platforms, C2 nodes, and communication nodes. A plethora system definitions may be included in this definition, but a basic example was expressed to allow for simplicity of explanation and analysis in this thesis. Futhermore, the systems can be further specified with value properties to represent vastly different systems with similar operations. For example, the same radar sensor block can represent anything from a ballistic missile tracking system to a radar for asteroid tracking to a hand-held radar used by a traffic officer. The capability for having a highly diversified set of systems exists within SysML.

6.2.7 Trans-domain Nature

SysML is designed to examine a system through various viewpoints. Relating back to the "Four Pillars of SysML" (see Chapter 1), the major focus of SysML is on the structure, behaviors, requirements, and parametrics of a system. These key viewpoints maintain when examining a SoS and require a trans-domain approach. The structural aspects of a SoS examine the hierarchical definition of the systems as well as the physical properties of those systems. With the use of CAD tools, further physical definition of each system can be provided in detail. Electric properties of the system could also be defined here or through a circuit diagram in SysML (often created with an *ibd*). Behavioral aspects of the SoS can be extended to require the use of external tools such as STK to capture the orbital dynamics of a satellite or simulation tools. Requirements engineering is captured entirely in SysML while parametrics may use SysML tools or external analysis tools (e.g., MATLAB).

Furthermore, the trans-domain nature of the SoS is established by the set of systems that are modeled and what the goal of the model is. It may be necessary to look at the data production and network usage of a SoS, and so these features of the systems and their interactions will be captured in SysML – as is done with the sensor model in Chapter 5. On the other hand, if the model is being built to examine software architectures or decision making protocols between systems, more focus may be placed on the operational aspects of the model. The model would function as a testbed for the integration of various systems' operational protocols. SysML has the capabilities to examine various different domains within a SoS, but it is still the duty of the practitioner(s) to develop the model for their needs.

6.2.8 Networks

Networks are common traits of SoS's, but not a necessary condition. It stems from the fact that systems are generally geographically distributed, yet must still interact. This interaction then occurs over a communication network. In SysML, communication networks are established through the use of flow ports and connections, usually shown on an *ibd*. Chapter 4 discusses methods for capturing system interactions through the use of networks, broken into physical and logical networks.

Table 6.2 Overview of SoS Traits in SysML

SoS Trait	**SysML Element/Diagram**
Managerial Independence	Composition/aggregation relation and/or stereotypes on blocks
Operational Independence	Activities allocated to blocks
Evolutionary Behavior	Top-level domain blocks to represent intermediate states
Emergent Properties	Activities, sequences, parametric diagrams, simulations
Geographic Distribution	Value properties
Heterogeneity	Unique blocks, value properties, instances
Trans-domain Nature	All views
Networks	Flow ports, connections, ibd

6.3 Is SysML capable of modeling different types of SoS?

Examples of three of the four types of SoS are developed throughout this thesis – directed, acknowledged, and collaborative. The virtual SoS is foregone due to the fact that there are no strict definitions on how systems interact, what high-level goals exist, or how to affect the evolution of the SoS. This abstract and free-form nature makes a rigorous SoSE approach to managing virtual SoS's an ineffective activity. A much more adaptive and free-form method to study virtual SoS's must be used and SysML would not be the tool to perform these analyses.

On the other end of the spectrum, directed SoS's lend themselves well to SysML modeling. This is due to the straight-forward definition of both hierarchical systems, how they interact, and the process by which they evolve. Similarly, acknowledged SoS's follow similar modeling practices but require the model designer to ensure that independence is stated between systems, both operationally and managerially. The primary difference between these two types is the existence of a high-level command that can influence systems in the directed SoS and the allocation of activities to systems.

Collaborative SoS's are a different story, as systems here have a voluntary inclusion into the SoS. A standard example of a collaborative SoS is the Internet. Capturing the entirety of a collaborative SoS may not be possible or desirable in SysML, chiefly due to the fact that systems may be included or removed at any point. More applicable to SysML is examining a specific subset or situation within a collaborative SoS. Modeling what happens when a new system joins or how multiple systems may interact is still possible, but the problem must be bounded. Also, by capturing the basic interactions between these systems, a detailed analysis or simulation script could extrapolate the problem set to a much larger scale without requiring additional modeling. The example shown in this thesis focuses on the interactions of a limited number of systems to focus on the rules and interactions between them, using a collaborative group of sensors interacting to track targets via a voluntary data sharing method. If a system decided or could not share any data, it would stop receiving data from the other sensors, possibly impairing its capability to track as many objects. Developing requirements and examining the effects of those requirements on how systems in this SoS interact is the best use of SysML for collaborative SoS's.

6.4 Future Work

As stated previously, one of the drawbacks of SysML is the time it takes to develop various diagrams. However, there are methods within SysML to create patterns and scripts to automate parts of this creation process. Some of these capabilities were used to do the automatic port and connection coloring in Chapter 5. Patterns could possibly be created to automate the rewiring of network via an adjacency matrix or to automatically connect system properties with constraint blocks. These patterns can be grouped into modules that are importable into any SysML model. By developing a standard ontological framework for modeling SoS's in SysML and then supporting that framework with patterns for developing common diagrams and scripts for automated analysis, the time to develop models can be reduced. This, however, is bound to be a daunting task that requires both a large amount of development, but also application to real world problems to verify and validate the processes.

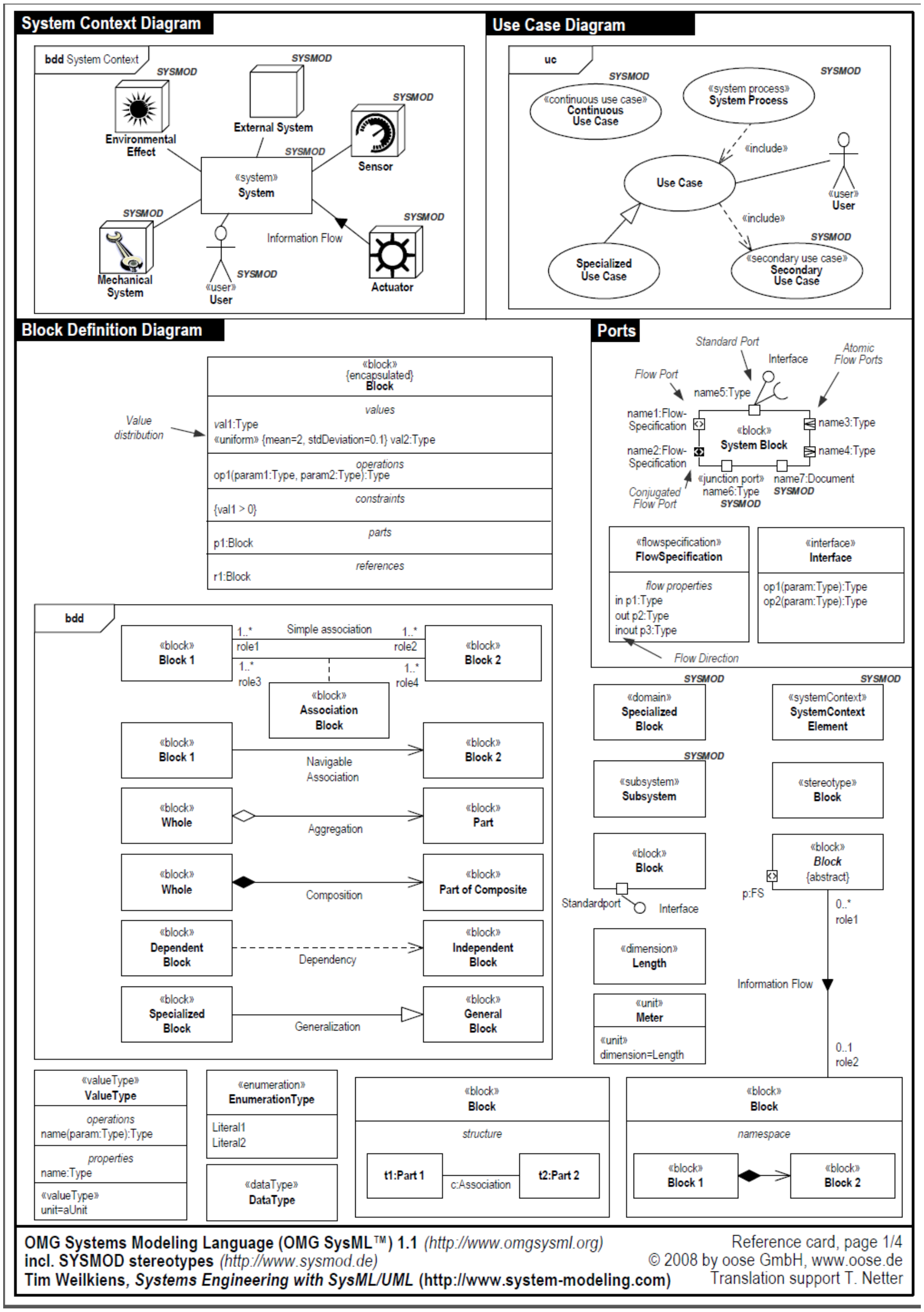
The development of an ontology for performing SoSE in SysML would help to both alleviate inconsistencies between the two and allow for future practitioners to more easily use these tools. This would require the definition of specific stereotypes, views, and patterns to be used. Though this research provides some of the patterns and views within SysML that are useful in regards to SoSE, it does not rigidly define a set of rules and processes. There are some frameworks that have laid the groundwork in this area and found acceptance in SysML (e.g. DoDAF, MoDAF), but they only detail a set of high-level viewpoints and the dependencies between these viewpoints. Nor is there a specific implementation of these frameworks within SysML.

Finally, work is ongoing in methods to strengthen the analysis capabilities of SysML. This is done by both providing plugins to interact with standard external analysis tools (CAD, CFD, etc.) and by improving tools to analyze the SysML elements themselves. As mentioned in Chapter 5, the Cameo Simulation Toolkit for MagicDraw allows for the direct analysis of activity diagrams but is still in its infancy of development. Furthermore, most scripting efforts are application-focused and limited in their scope. Improved analysis capabilities within tools should be developed in order to improve the functionality of SysML, as this is generally one of the limiting factors to its adoption.

APPENDICES

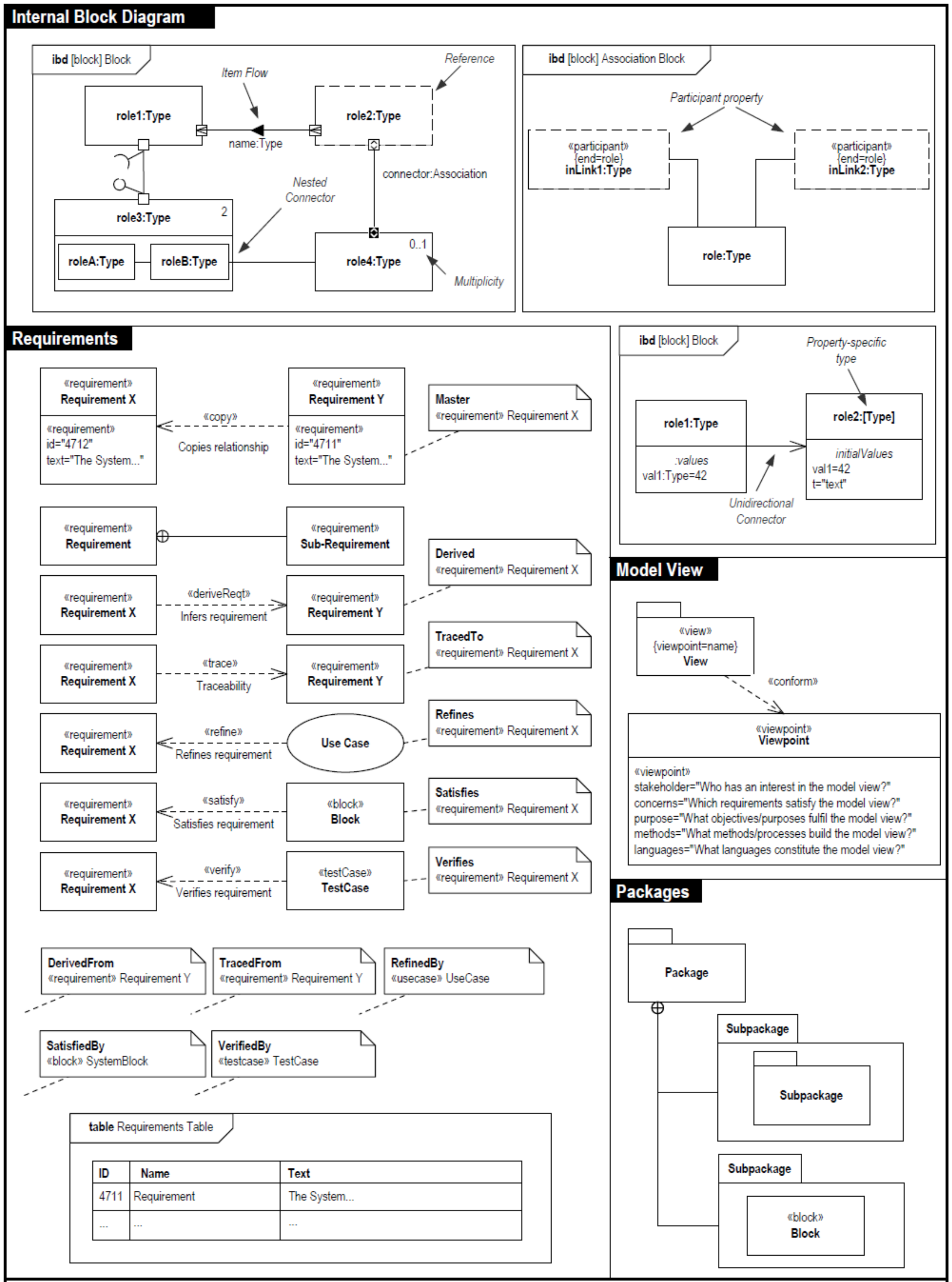
A. SYSML TUTORIAL

The following is a set of reference materials displaying most standard SysML elements and interactions.



OMG Systems Modeling Language (OMG SysML™) 1.1 *(http://www.omgsysml.org)*
incl. SYSMOD stereotypes *(http://www.sysmod.de)*
Tim Weilkiens, *Systems Engineering with SysML/UML* (http://www.system-modeling.com)

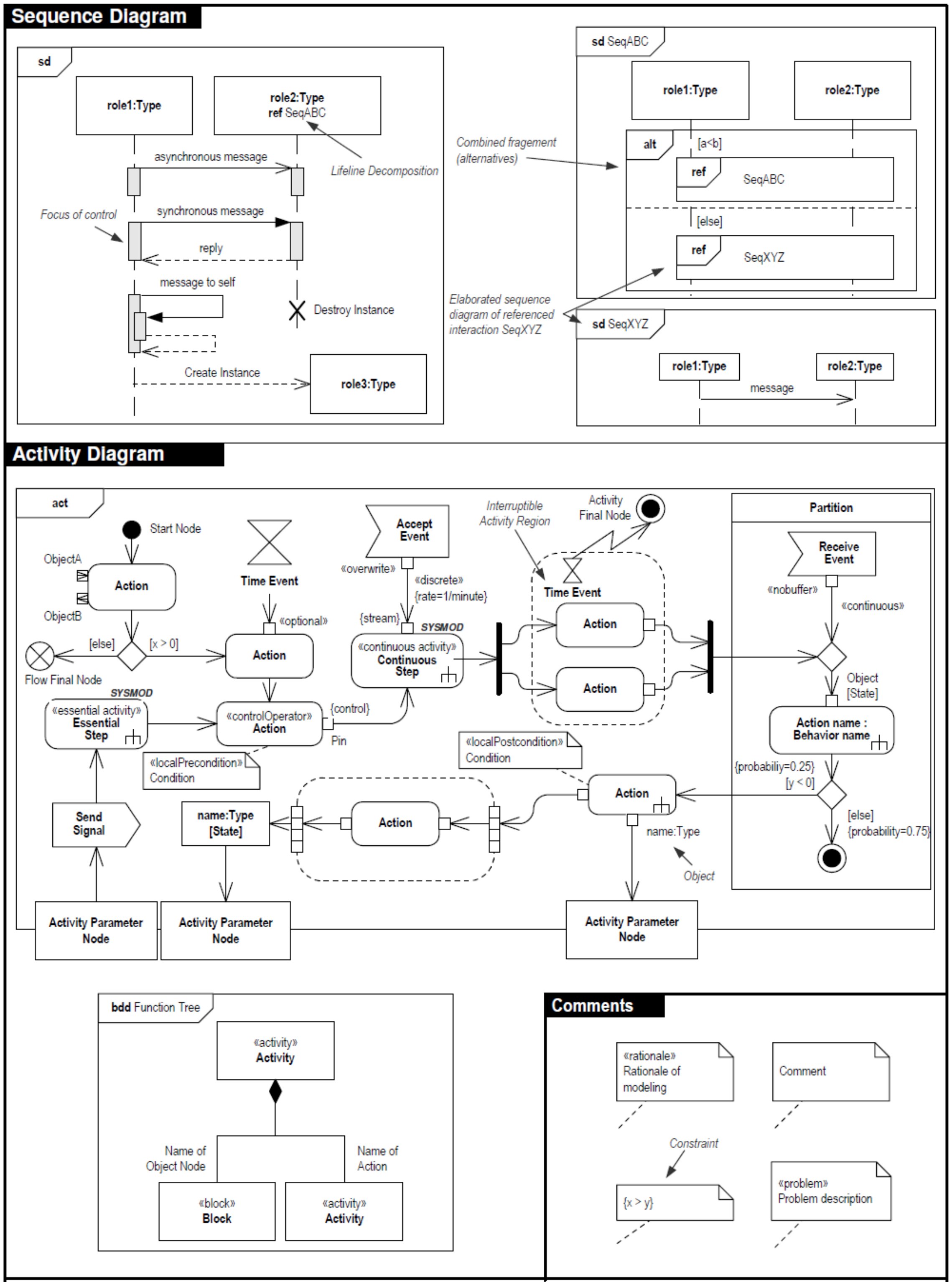
Reference card, page 1/4
© 2008 by oose GmbH, www.oose.de
Translation support T. Netter


Internal Block Diagram
ibd [block] Block
Item Flow
Reference
role1:Type
role2:Type
name:Type
connector:Association
Nested Connector
role3:Type
2
roleA:Type
roleB:Type
role4:Type
0..1
Multiplicity
ibd [block] Association Block
Participant property
«participant» {end=role} inLink1:Type
«participant» {end=role} inLink2:Type
role:Type
Requirements
«requirement» Requirement X
«requirement» id="4712" text="The System..."
«copy»
Copies relationship
«requirement» Requirement Y
«requirement» id="4711" text="The System..."
Master «requirement» Requirement X
«requirement» Requirement
«requirement» Sub-Requirement
Derived «requirement» Requirement X
«deriveReqt»
Infers requirement
TracedTo «requirement» Requirement X
«trace»
Traceability
Refines «requirement» Requirement X
«refine»
Refines requirement
Use Case
«satisfy»
Satisfies requirement
«block» Block
Satisfies «requirement» Requirement X
«verify»
Verifies requirement
«testCase» TestCase
Verifies «requirement» Requirement X
DerivedFrom «requirement» Requirement Y
TracedFrom «requirement» Requirement Y
RefinedBy «usecase» UseCase
SatisfiedBy «block» SystemBlock
VerifiedBy «testcase» TestCase
table Requirements Table
ibd [block] Block
Property-specific type
role1:Type
:values val1:Type=42
role2:[Type]
initialValues val1=42 t="text"
Unidirectional Connector
Model View
«view» {viewpoint=name} View
«conform»
«viewpoint» Viewpoint
«viewpoint»
stakeholder="Who has an interest in the model view?"
concerns="Which requirements satisfy the model view?"
purpose="What objectives/purposes fulfil the model view?"
methods="What methods/processes build the model view?"
languages="What languages constitute the model view?"
Packages
Package
Subpackage
Subpackage
Subpackage
«block» Block

ID	Name	Text
4711	Requirement	The System...
...	...	...

OMG Systems Modeling Language (OMG SysML™) 1.1 (http://www.omgsysml.org)
incl. SYSMOD stereotypes (http://www.sysmod.de)
Tim Weilkiens, Systems Engineering with SysML/UML (http://www.system-modeling.com)

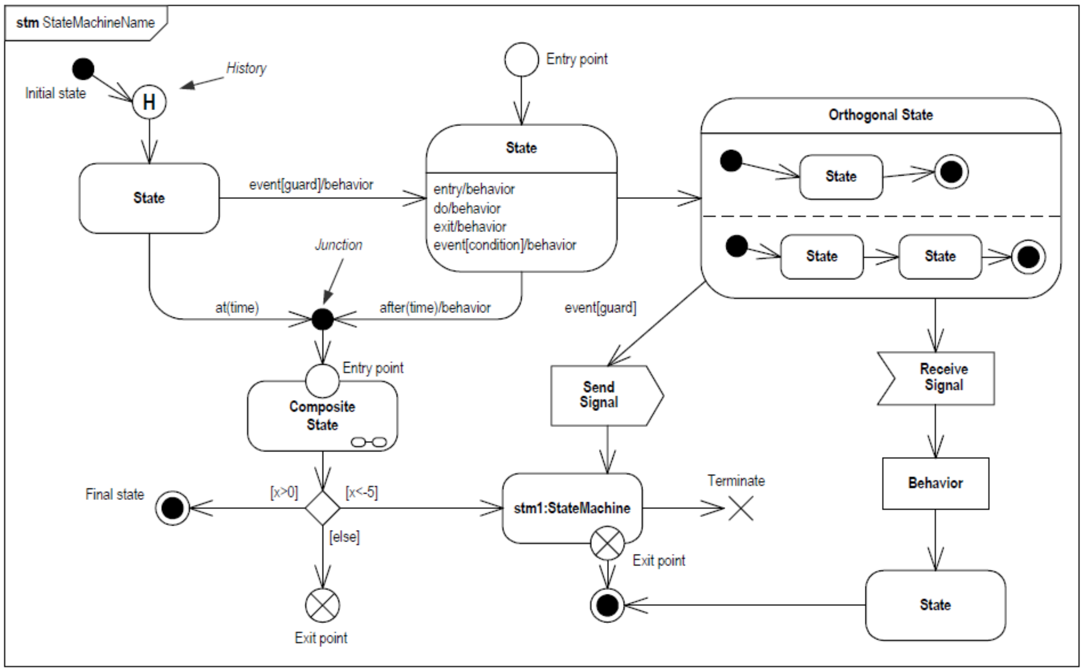
Reference card, page 2/4
© 2008 by oose GmbH, www.oose.de
Translation support T. Netter



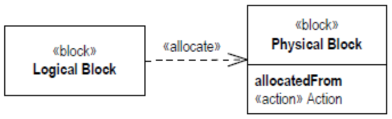
OMG Systems Modeling Language (OMG SysML™) 1.1 (http://www.omgsysml.org)
incl. SYSMOD stereotypes (http://www.sysmod.de)
Tim Weilkiens, *Systems Engineering with SysML/UML* (http://www.system-modeling.com)

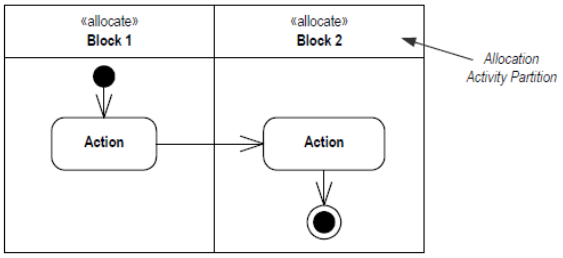
Reference card, page 3/4
© 2008 by oose GmbH, www.oose.de
Translation support T. Netter

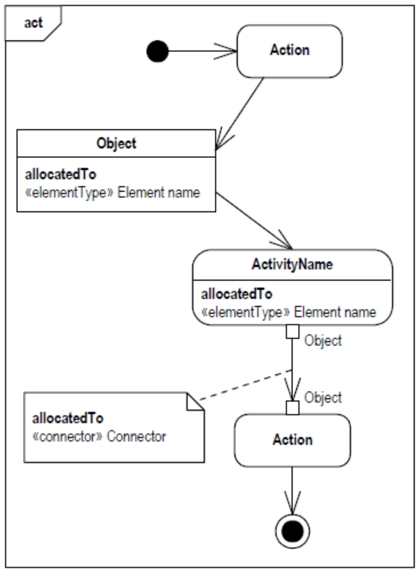
State Machines



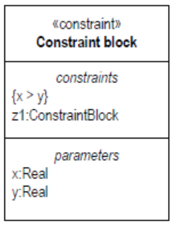
Allocations

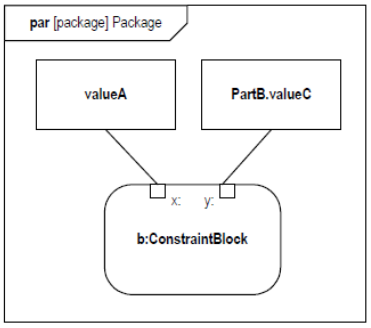






Parametric Diagram





B. MODELING PATTERN

This appendix describes in detail the pattern used throughout the modeling effort for this thesis. Key aspects of the pattern are described throughout the main chapters and are further elucidated here. Alternative model design options are discussed along with explanations on why these options were not used. While the main chapters of this thesis focus on the SoS engineering aspects of SysML, this appendix will describe the nuances and usage of SysML itself.

B.1 Developing the Baseline Architecture and Specialized Configurations

The baseline architecture is initialized with a single context block, used to house all components of the architecture. This is required in order to provide a location for global properties while also acting as a method for generalizing any configurations of the architecture. In the baseline hierarchy block definition diagram (Figure B.1), the *SensorArchitecture* block acts as the context block. Any elements common to all configurations are housed within this context block, while elements unique to a configuration may add to or redefine the common set (these specific relations are discussed later in this Chapter).

Components of the architecture are instantiated as blocks and related to the context blocks via directed associations, or composition relations. The multiplicity of elements may be specified within this relation to constrain the number of elements designated for either end of the relation. From Figure 3.1, the *SensorArchitecture* is composed of one-to-any number of **ControlGroup**s, zero-to-any number of *CommNode*s, zero-to-one *HighLevelCommand*s, and zero-to-any number of *Target*s. The internal elements of each of these blocks is further defined through composition relations.

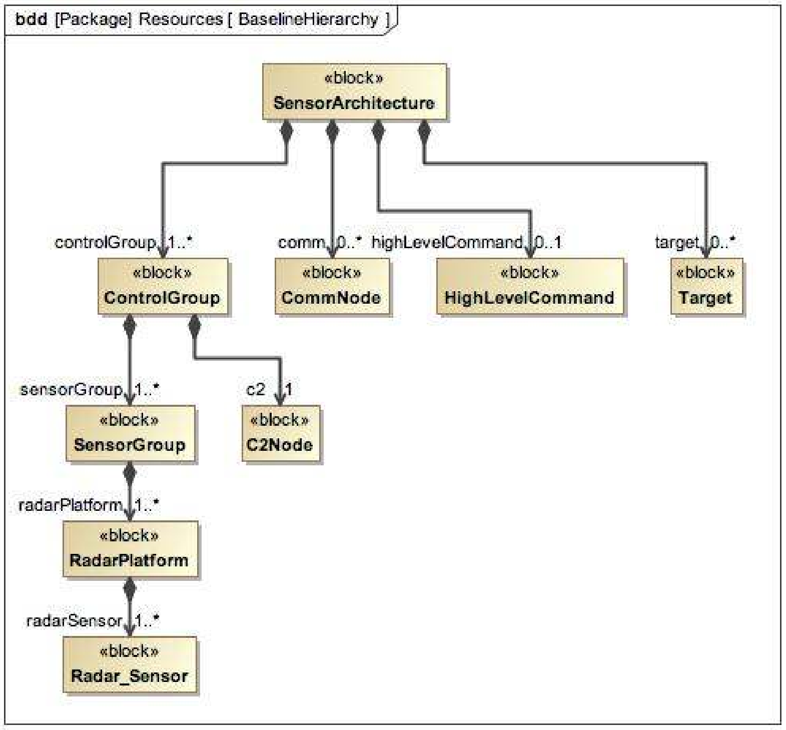


Figure B.1. Block definition diagram showing the baseline hierarchy of systems within the SoS

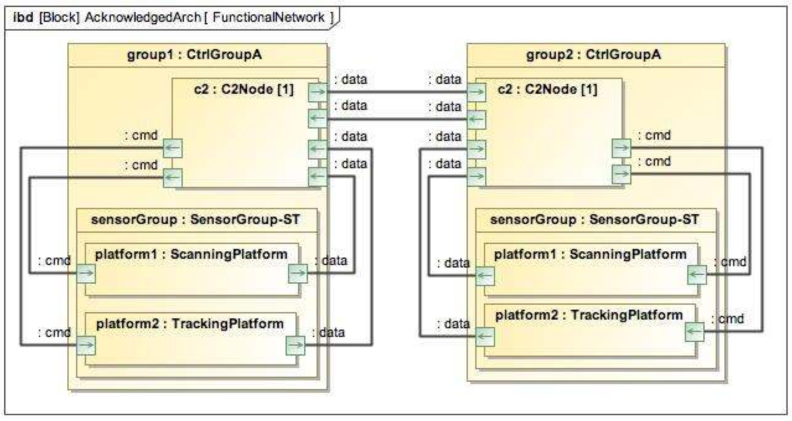

ibd [Block] AcknowledgedArch [FunctionalNetwork]
group1 : CtrlGroupA
c2 : C2Node [1]
: data
: cmd
sensorGroup : SensorGroup-ST
platform1 : ScanningPlatform
platform2 : TrackingPlatform
group2 : CtrlGroupA
: cmd
: data

Figure B.2. Internal Block Diagram showing the interactions between internal elements

Structuring baseline architectures in this way allows for scalability and compartmentalizes the system definition. This scalability allows for further definition in either direction (higher or lower level) with ease. To increase the scale of the model, a new context block is defined with a directed association to the previous context block and any other elements at that level. To increase the fidelity of the model, more detail is provided by defining the composition of existing elements. This capability is crucial to SoS Engineering where defining the scale of the model may change as the SoS evolves, requirements change, or higher fidelity analysis is required.

B.1.1 The use of directed associations

Directed associations are useful in SysML for specifying a structural hierarchy of elements. By default, a directed aggregation will create a part property within the relation-supplying block. Part properties may be displayed on any Internal Block Diagrams (ibd's) belonging to the owning block. Whereas block definition diagrams are used to describe the hierarchy of elements in an architecture, an ibd is used to show the interactions of parts within a block.

An example ibd can be seen in Figure B.2, which shows the logical interactions for the Acknowledged SoS. ibd's only display the part properties within a block; thus, all parts on the diagram are owned by the type of the part within which it is nested.

A key aspect of the directed association is that it establishes a common composition for all blocks of a similar type. This may seem useful in the case of duplication of systems or components; however, it provides no benefits on its own for architectures in which systems may differ slightly. To best model these discrepancies a more sophisticated approach must be used – implementing directed associations, generalizations, and redefinition.

B.1.2 Applying generalizations and redefinition

Generalization relations allow for common attributes to be inherited by specific blocks. Using these generalization relations, common attributes are shared. The block inheriting these shared attributes is deemed the specialized block.

As the baseline architecture is created from the generalized blocks, a specialized configuration of this architecture then redefines any blocks that specialize the blocks that compose it. For example, a baseline architecture may be composed of multiple general radar platforms – non-destinct in their detail. In order to specify these as certain types of radar and provide detail to their composition, which may vary among each system, specialized blocks for each radar may redefine them under the specialized architecture's context. Furthermore, any ancestors (parents and their parents) of these redefined blocks must also be redefined up to the redefined context architecture block.

This pattern for defining an architecture allows for a taxonomical definition of a SoS and its components, while the specialized architecture demonstrates a specific usage or deployment of systems meeting that definition. An alternate method for meeting this need is to use SysML "instances" of the original blocks.

Instances are highly limited in their usage. They can only specify detail on existing properties of the blocks that they instance. Therefore, no additional detail or design can be included in an instance. For example, an instance of a radar platform from the baseline architecture is incapable of providing further design details outside of altering the base values of any properties (e.g., changing operational bandwidth or power), whereas a redefinition of that block may utilize the full capabilities of SysML to define subsystems that hold these values, to create new relations to activities that the system performs, and/or to satisfy new requirements not levied on the baseline system.

B.2 Definition of System Interactions

In order to capture interactions between systems, SysML ports, connectors, interfaces, and association blocks are utilized. Ports act as an element on a block (note: not a part property) that acts as an entry and/or exit point. Connectors allow these ports to be linked. Interfaces are used to type ports and provide details as to their attributes. Finally, association blocks link interfaces and detail their interaction.

Together these elements define the possible and enacted interactions between systems within an architecture. When a port is placed on block, it establishes a gateway with which that block may interact. That port is then typed by an interface, which declares how that port acts. These interfaces are defined in the same way as blocks through the use of composition and generalization relations – allowing for an ethernet interface to inherit the properties of a generic data link layer interface, or for a high-level logical command interface to be composed of various ethernet interfaces.

Once the ports are defined, they are linked through the use of connections. These connections are then further defined through the use of association blocks. Similar to how interfaces define ports, association blocks are used to type and provide detail to connectors, and can also capitalize on composition and generalization relations. Furthermore, they allow for internal validation of the connector. For a connector to be valid, its association block must link the interfaces or instances of the interfaces that type the ports on either end of the connector. For example, two ethernet ports cannot be connected with a coaxial cable, as this connector and its association block type (coaxial) would not be compatible with the ethernet interfaces that type the ports on either end of the connector.

An alternative method for defining interactions over connectors is the use of conveyed information flows as opposed to association blocks. This method is also uni- and bi-directional, allows for typed connectors and ports, and can capitalize on composition and generalization relations. A boon of using conveyed information flows is that multiple flows may be applied to a single connector, which is useful for high-level logical interactions, saving the time of modeling multiple connectors. However, conveyed information flows do not require relations to the ports' interfaces, so the inherent validation within the model is lost. As these methods both have nearly identical capabilities, it does not impact any of the points made in this thesis on which is used; neither limit the capabilities of SysML for performing SoSE tasks.

LIST OF REFERENCES

LIST OF REFERENCES

[1] D. M. Buede, *The Engineering Design of Systems, Models, and Methods.* Wiley, 2009.

[2] E. Rebentisch, D. H. Rhodes, and E. Murman, “Lean systems and engineering: Research and initiatives in and support of a new and paradigm,” tech. rep., Conference on Systems Engineering Research, 77 Massachusetts Ave., Bldg 41-205 Cambridge, MA 02139, Apr. 2004.

[3] M. W. Maier, “Architecting principles for systems-of-systems,” in *Sixth Annual International Symposium of INCOSE*, (Boston, MA), 1996.

[4] M. Jamshidi, ed., *System of Systems Engineering: Innovations for the 21st Century.* Wiley Series in Systems Engineering and Management, Wiley, 2009.

[5] S. Jenkins, “A modeling approach to document generation,” in *INCOSE Insight*, INCOSE, 2009.

[6] A. L. Ramos, J. V. Ferreira, and J. Barcelo, “Model-based systems engineering: An emerging approach for modern systems,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, pp. 101–111, Jan. 2012.

[7] D. A. DeLaurentis, “Understanding transportation and as system-of-systems and design and problem,” tech. rep., AIAA, Jan. 2005.

[8] M. W. Maier, “Research challenges for systems-of-systems,” in *IEEE International Conference on Systems, Man, and Cybernetics*, Oct. 2005.

[9] O. of the Deputy Under Secretary of Defense for Acquisition, S. Technology, and S. Engineering, *Systems Engineering Guide for Systems of Systems.* ODUSD(A&T)SSE, version 1.0 ed., 2008.

[10] R. Peak, “Sysml parametrics research for modeling & simulation interoperability,” tech. rep., Modeling & Simulation Lab, Georgia Institute of Technology, 2011.

[11] M. Horl, M. Hochwallner, S. Dierneder, and R. Scheidl, “Integration of sysml and simulation models for mechatronic systems,” in *EUROCAST 2011*, 2, pp. 89–96, ASME, Sept. 2011.

[12] “Unified modeling language.” http://www.uml.org/, Apr. 2013.

[13] “Paramagic plugin.” http://www.nomagic.com/products/magicdraw-addons/paramagic-plugin.html, 2013.

[14] “Omg systems modeling language.” http://www.omgsysml.org/, 2013.

[15] J. Dahmann, G. Rebovich, R. Lowry, J. Lane, and K. Baldwin, "An implementers view of systems engineering for systems of systems," tech. rep., IEEE, 2011.

[16] B. S. Blanchard and W. J. Fabrycky, *Systems Engineering and Analysis*. Prentice Hall, 2010.

[17] D. A. U. Press, *Systems Engineering Fundamentals*. Department of Defense, Systems Management College, Fort Belvoir, VA, Jan 2001.

[18] J. A. Estefan, ed., *INCOSE Survey of MBSE Methodologies*. INCOSE, May 2007.

[19] J. A. Lane and T. Bohn, "Using sysml modeling to understand and evolve systems of systems," in *Systems Engineering*, vol. Vol 16., pp. pp. 87–98, Wiley Periodicals, Inc., Apr. 2013.

[20] S. M. White, "Modeling a system of systems to analyze requirements," in *IEEE International Systems Conference*, (Vancouver, Canada), IEEE, IEEE, March 2009.

[21] D. A. DeLaurentis, W. A. Crossley, and M. Mane, "Taxonomy to guide systems-of-systems decision-making in air transportation problems," *Journal of Aircraft*, vol. 48, pp. 760–770, May 2011.

[22] J. A. Lane, "Factors influencing and system-of-systems and architecting and integration and costs," in *Conference on Systems Engineering Research*, Stevens Institute of Technology, March 2005.

[23] M. Liotine, *Mission-Critical Network Planning*. Artech House, 2003.

[24] "Space surveillance sensors: The pave paws and bmews radars." http://mostlymissiledefense.com/2012/04/12/pave-paws-and-bmews-radars-april-12-2012/, Apr. 2012.

[25] S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*. Artech House, 1999.

[26] R. Peak, R. Burkhart, S. Friedenthal, M. Wilson, M. Bajaj, and I. Kim, "Simulation-based design and using sysml and part 2: Celebrating diversity and by example," in *INCOSE Intl. Symposium*, (San Diego), INCOSE, 2007.

[27] R. K. Garrett, S. Anderson, N. T. Baron, and J. D. Moreland, "Managing the interstitials, a system of systems framework suited for the ballistic missile defense system," *Systems Engineering*, vol. 14, pp. 87–109, Mar 2011.